Finding Your Way on the Map

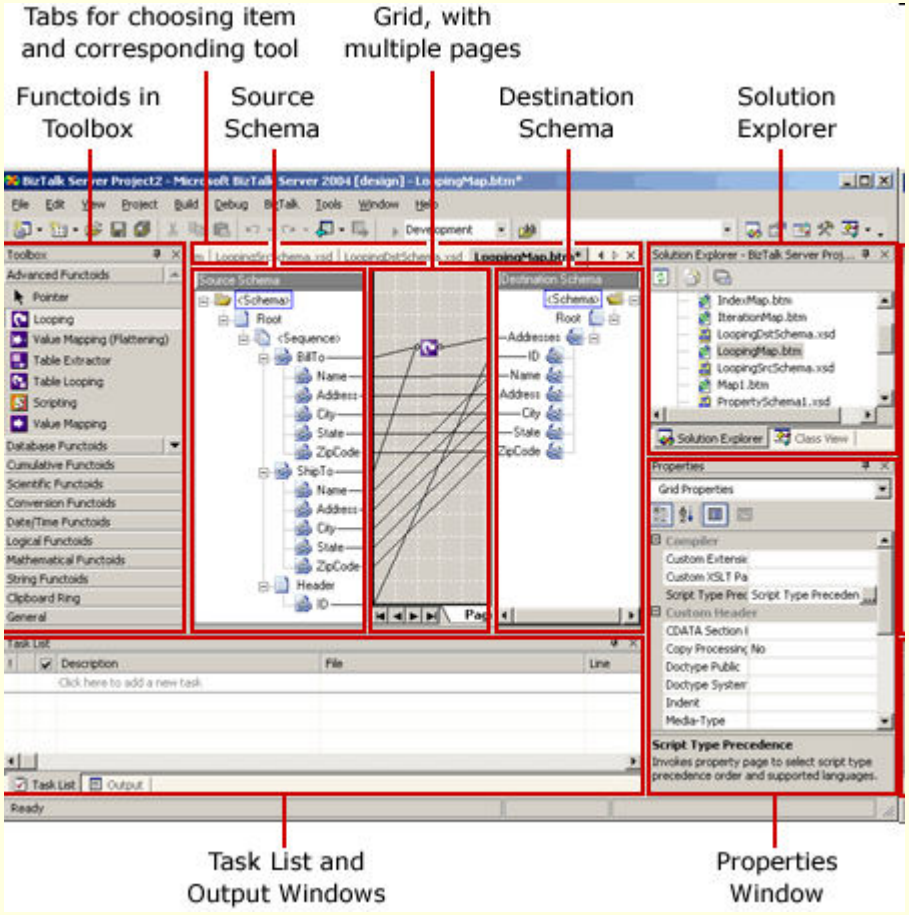
Using the BizTalk Mapper

Rupesh Chintapalli (MCTS)

BizTalk Mapper Basics

- Maps elements from one schema to another
- Lives within BizTalk projects in the Visual Studio shell
  - Source schema tree view
  - Destination schema tree view
  - Grid view (links and functoids workspace)
  - Toolbox window (drag-and-drop functoids)
  - Properties, Solution Explorer, Task List, etc
- Maps have a *.btm extension
- Use UTF-16 encoding
- Generates XSLT

BizTalk Mapper Basics



Input / Output

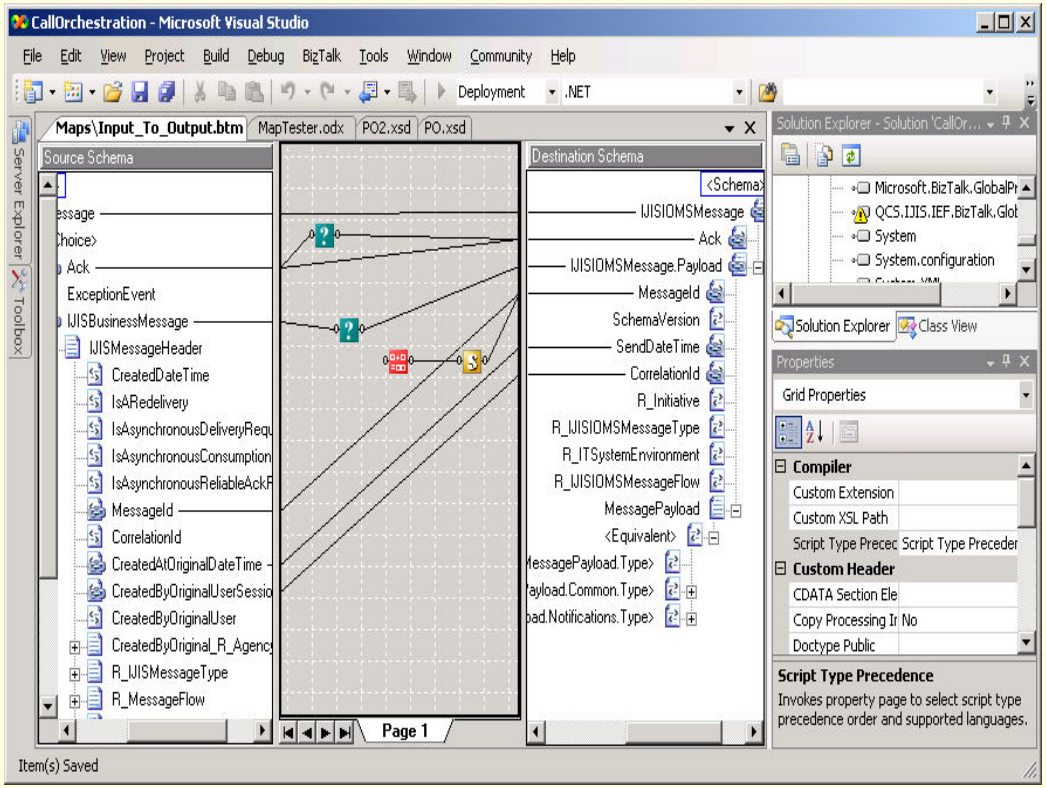
- Select target & destination schema from within the Mapper GUI itself
- Displays schema in read-only tree-view (ala BizTalk Schema Editor)
- Many-to-One, etc can be mapped, but map needs to be created from within a BizTalk orchestration
  - Use the Transform shape to select the target & destination messages
  - Select option to create a new map
  - Cannot be hosted in a send/receive port

The Grid

- Workspace for drawing links between source & target nodes
- Can incorporate one or more functoids within the link path
- Scrollable – larger area than visible all at once
- Grid can be divided into multiple pages to organise large/complex maps in logical sub-divisions

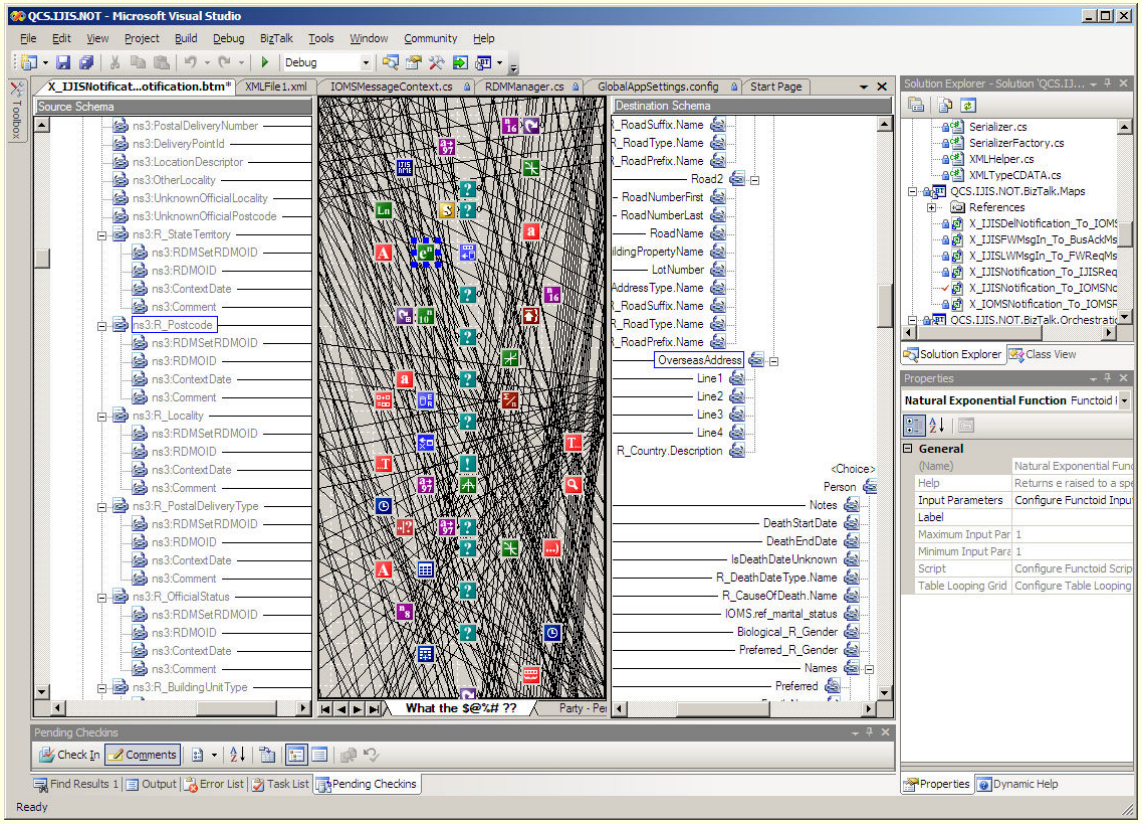
The Grid – Multiple Pages

- Maybe not so important here…



The Grid – Multiple Pages

- …*now* it's important!



Functoids

- BizTalk includes at least **80** functoids OOTB to perform a variety of XSLT operations:
  - String manipulation *(10)*
  - Mathematical operations *(11)*
  - Logical operations *(14)*
  - Conversions *(4)*
  - Database lookups *(10)*
  - Aggregation / Iteration / Flattening *(10+)*
  - Custom scripts
  - etc

Functoids

- Most functoids take one or more input parameters
  - Can be nodes from input schema, other functoids, or constants
  - Default values are 0 for numeric parameters, blank for others
- All functoids output data into the mapped output element
  - Can be XML or simple data depending on the operation
  - When target is a record, set Mixed property = true
- Functoids may be chained
  - Always execute left to right

Scripting Functoid

- Allows you to execute custom code or XSLT
- Can call an external .NET assembly
  - Must be thread safe
  - Must be in the GAC
  - Classes/methods cannot be static
- Can execute custom script
  - C# .NET
  - VB.NET
  - JScript.NET
  - XSLT
  - XSLT Call Template

Custom Functoids

- Reasons to develop custom functoids:
  - Special validation/conversion rules that require a proprietary legacy API
  - Need to execute same custom script in multiple maps
  - Need to write to the event log
  - Need to generate hash code for use elsewhere
  - Need to encrypt/decrypt fields using custom logic

Developing Custom Functoids

1. Create a new class library project using the .NET language of your choice.
2. Using the strong-naming utility sn.exe, create a keyfile and assign it to the project.
3. Add a reference to **Microsoft.BizTalk.BaseFunctoids.dll**. This assembly contains the **BaseFunctoid** base class.
4. Create a resource file and add it to the project. Add string resources for the functoid name, tooltip, and description. Add a 16x16-pixel image resource to represent the functoid on the map designer palette.
5. Implement the functoid class by deriving from **BaseFunctoid**, establishing basic parameters in the constructor, and then writing the functoid method and any supporting methods. The assembly can contain multiple custom functoids.
6. Deploy the assembly to the GAC and the Mapper Extensions folder, and ensure the new functoid is available from the Toolbox palette.

Developing Custom Functoids

```
using System;
using Microsoft.BizTalk.BaseFunctoids;
using System.Reflection;

namespace SampleCustomFunctoid
{
     [Serializable]
     public class EncodeFunctoid : BaseFunctoid
     {
          public EncodeFunctoid() : base()
          {
               //Custom functoids should begin with 6000 or higher
               this.ID = 6667;

               // resource assembly reference
               SetupResourceAssembly
                    ("SampleCustomFunctoid.SampleCustomFunctoidResource",
                    Assembly.GetExecutingAssembly());

               //Set the properties for this functoid
               SetName("SAMPLECUSTOMFUNCTOID_NAME");
               SetTooltip("SAMPLECUSTOMFUNCTOID_TOOLTIP");
               SetDescription("SAMPLECUSTOMFUNCTOID_DESCRIPTION");
               SetBitmap("SAMPLECUSTOMFUNCTOID_BITMAP");
```

Developing Custom Functoids

```
        // one parameter in, one parameter out
        this.SetMinParams(1);
        this.SetMaxParams(1);

        //Function code below
        SetExternalFunctionName(GetType().Assembly.FullName,
        "SampleCustomFunctoid.EncodeFunctoid", "EncodeChars");

        //Category in Toolbox where this functoid will appear
        this.Category = FunctoidCategory.String;

        //output of functoid can go to all nodes indicated
        this.OutputConnectionType = ConnectionType.All;

        // add one of the following lines of code for every input
        // parameter. All lines would be identical.
        AddInputConnectionType(ConnectionType.All);
    }
```

Developing Custom Functoids

```
        // Actual function which does the replacement of
           symbols
        public string EncodeChars(String strInputValue)
        {
            strInputValue =
               strInputValue.Replace("&","&amp;");
            strInputValue =
               strInputValue.Replace("<","&lt;");
            strInputValue =
               strInputValue.Replace(">","&gt;");
            return strInputValue;
        }
    }
}
```

When All Else Fails…

- Use custom XSLT in place of a map:
  1. Create an empty BizTalk map in your project and set the source and destination schemas.
  2. In the Grid view, click the mapper grid.
  3. In the Properties window, select **Custom XSLT Path** and click the ellipsis (**…**) button.
  4. In the **Select Custom XSLT File** dialog box, navigate to the XSLT file and click the **Open** button.
- Requires XSLT 1.0 (XSLT 2.0 not supported)
- Custom extension XML file required if using external .NET assemblies

Testing Your Map

- Can test your map independently in VS without deploying it
- Set map properties:
  - Source schema
  - Output file *(optional)*
  - Option to validate input / output
- Right-click map and choose “Test Map”
- Output file saved

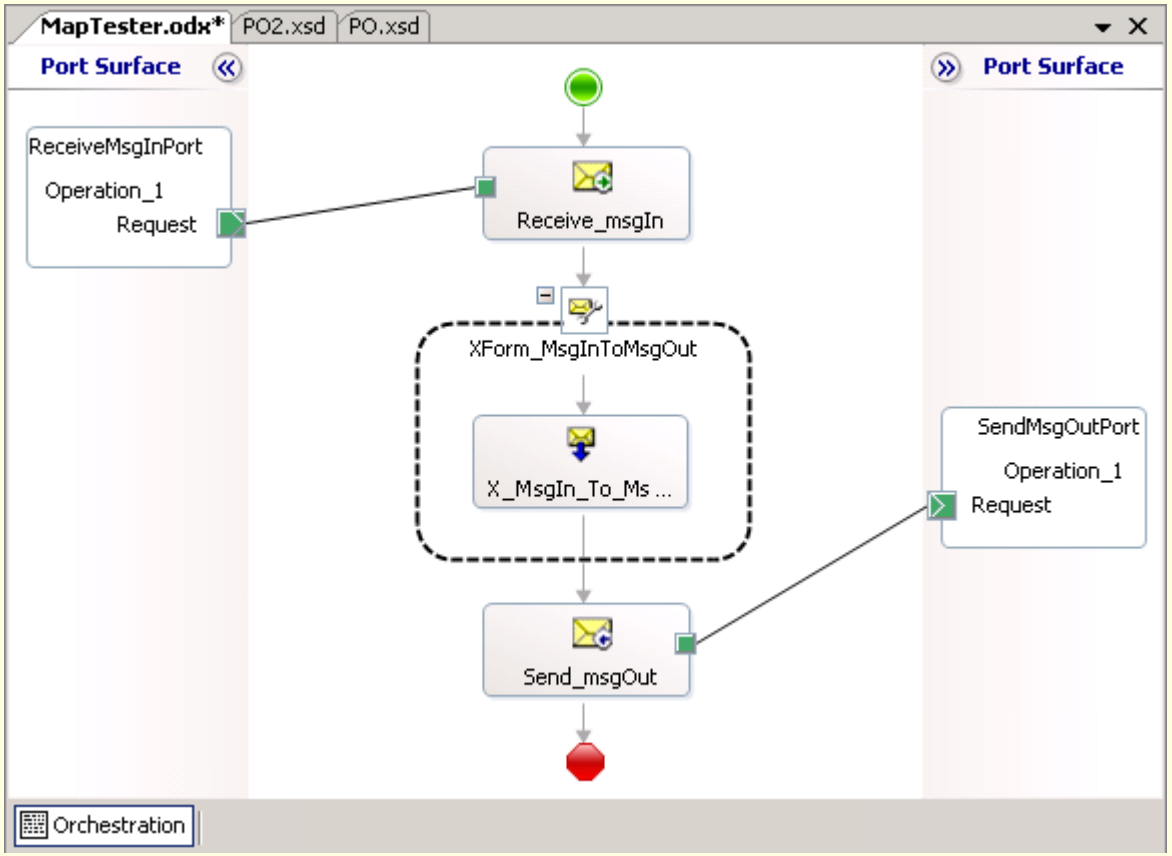
Default location:
$\Documents and Settings\[current user]\Local Settings\Temp_MapData

Reviewing the XSLT

- Inspecting the XSLT generated by the compiler provides insight into how the map functions
- Also provides another debugging option
- In Solution Explorer, right-click *.btm file and select “Validate Map”
- Link to generated XSLT shown in Output window

Hosting Your Maps

- Does your orchestration look like this?



Hosting Your Maps

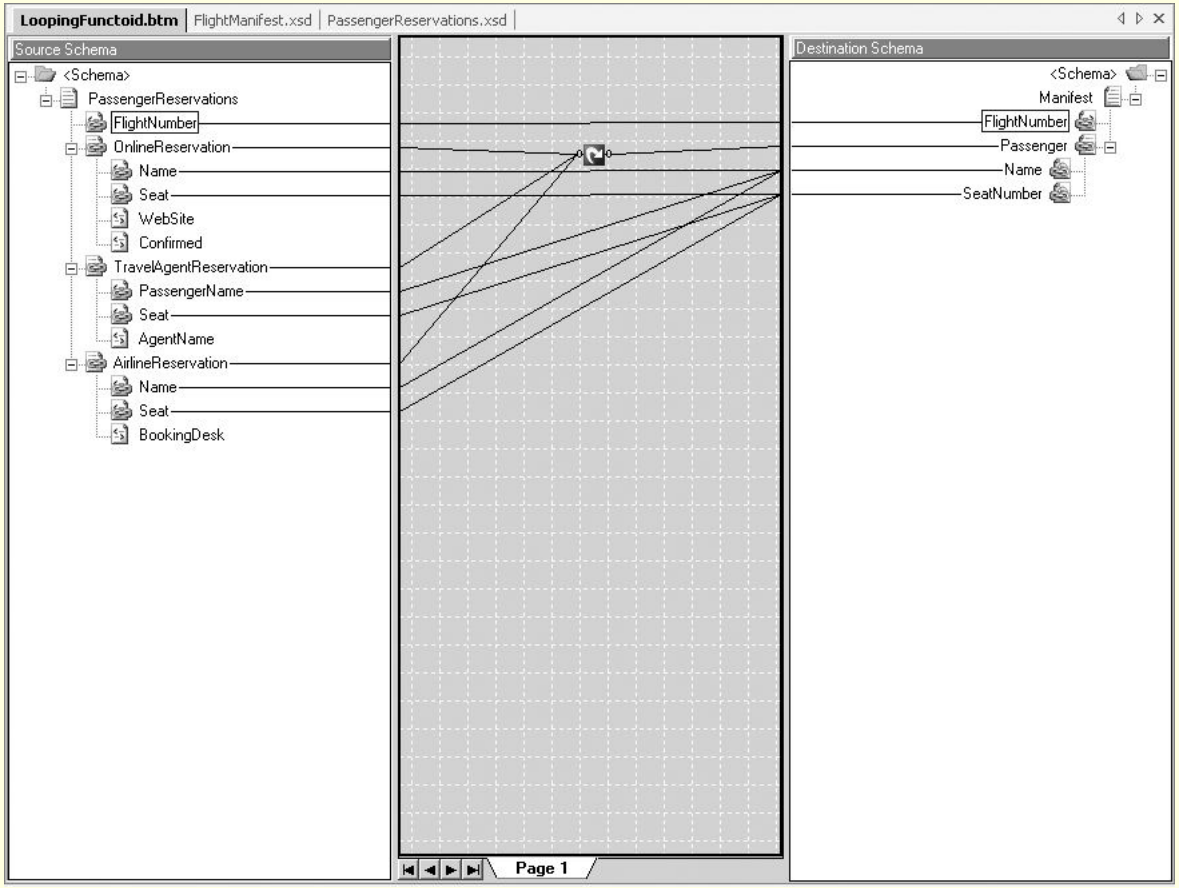
- One-to-One Mapping? Host it in a port!
- Don’t need an orchestration unless…
  - Mapping many-to-one
  - Logical branching required

Behind the Scenes

- Open *.btm file as an XML file
- Entire map represented as XML
- Edit schemas, links & namespaces in text
- Access to hidden attributes:
  - GenerateFixedNodes
  - OptimizeValueMapping
  - PreserveSequenceOrder
  - TreatElementsAsRecords

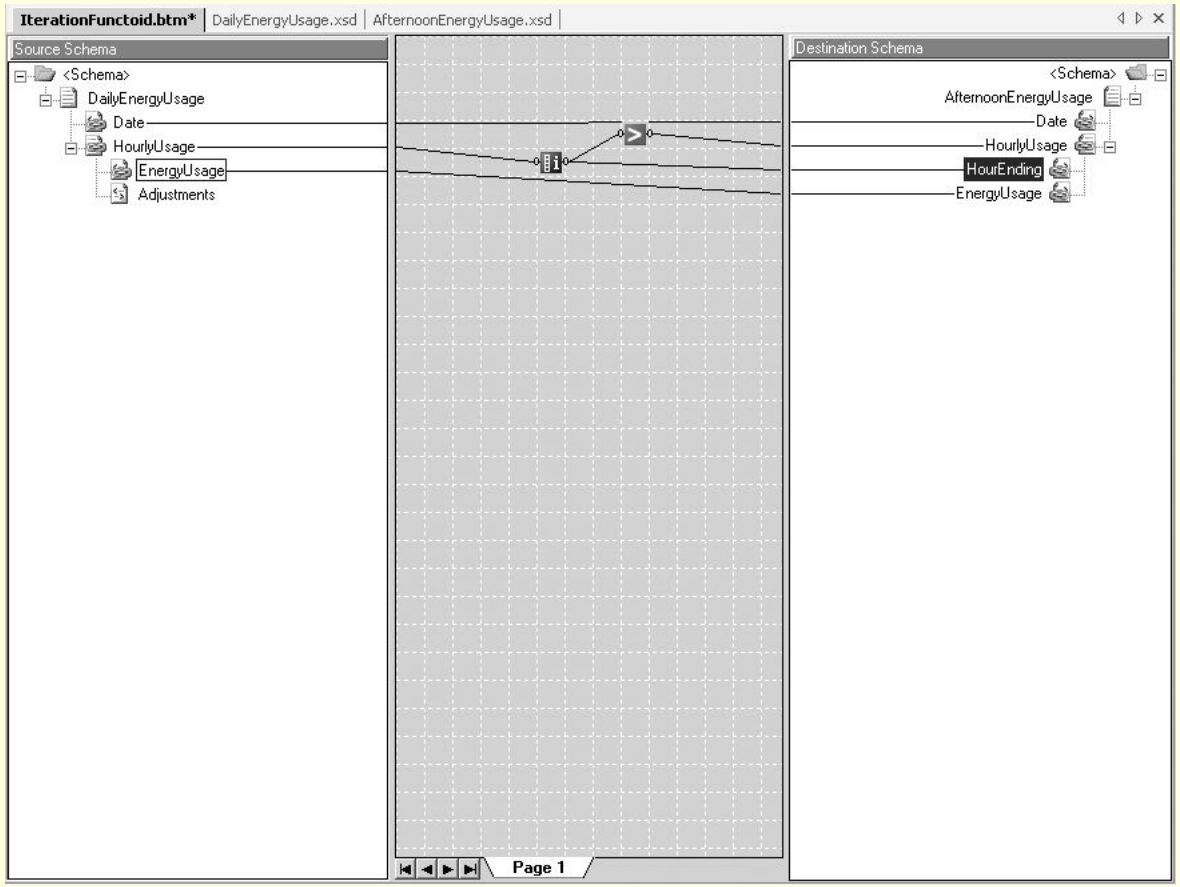
Common Mapping Logic Constructs

- Looping:



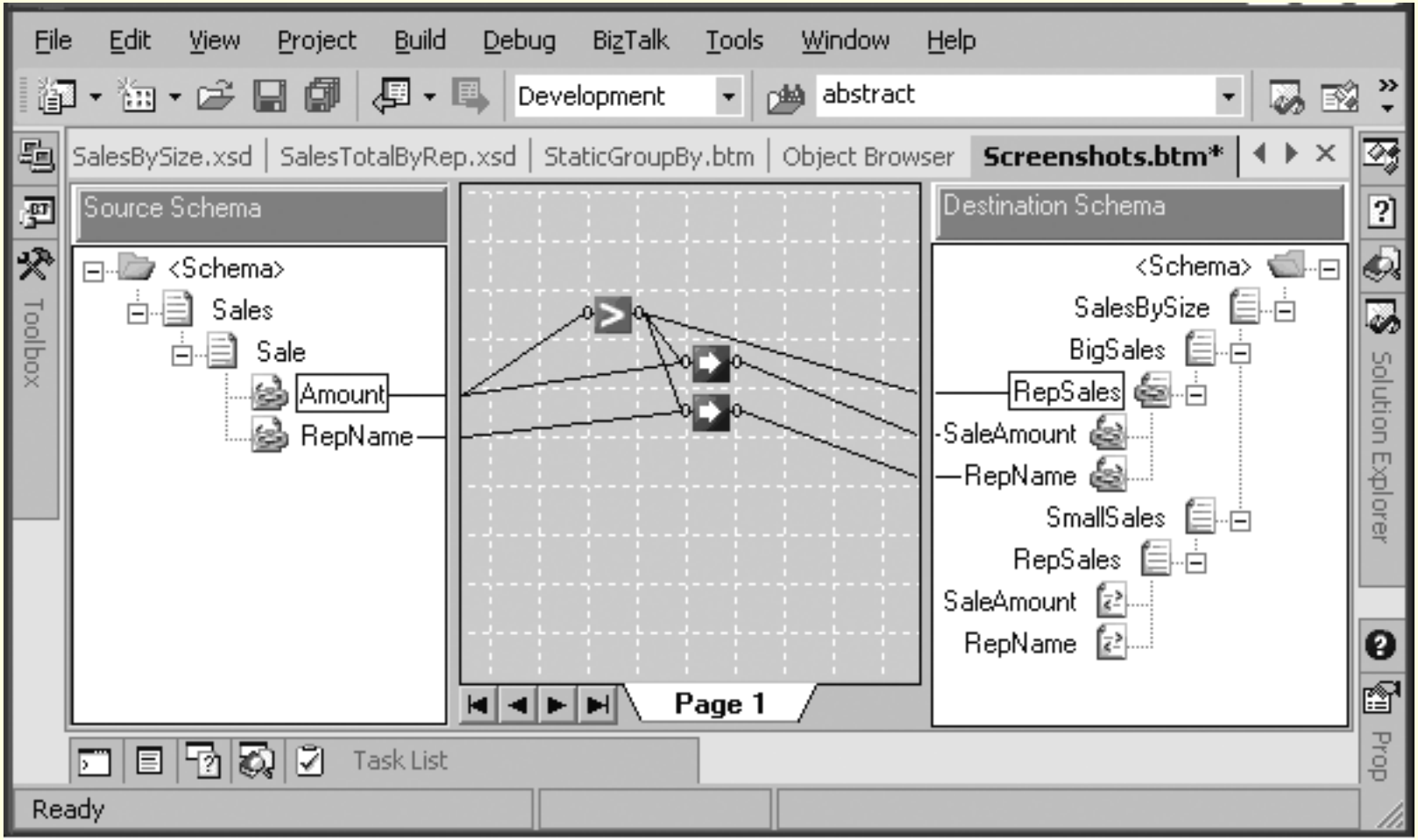
Common Mapping Logic Constructs

- Iteration:



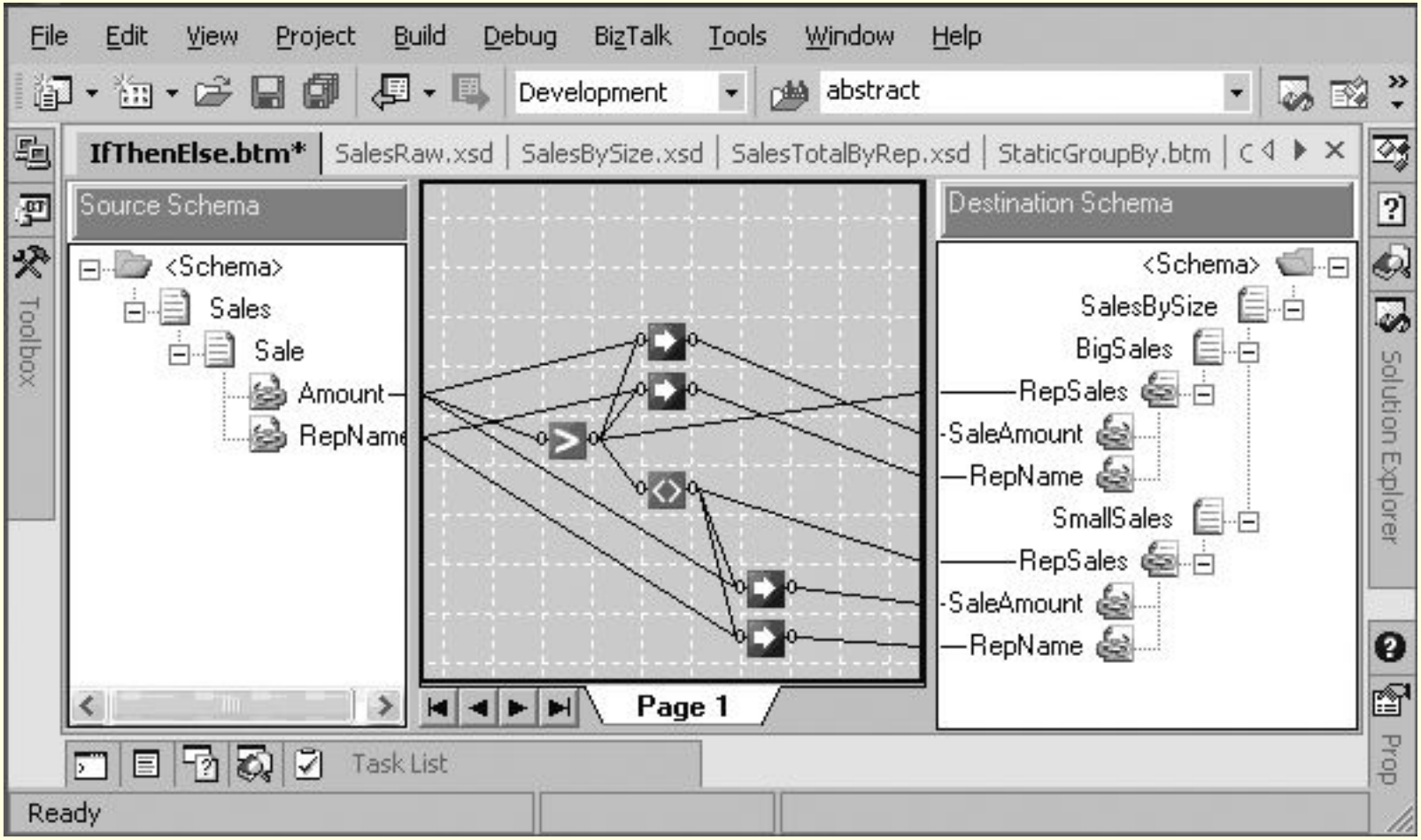
Common Mapping Logic Constructs

- If-Then:



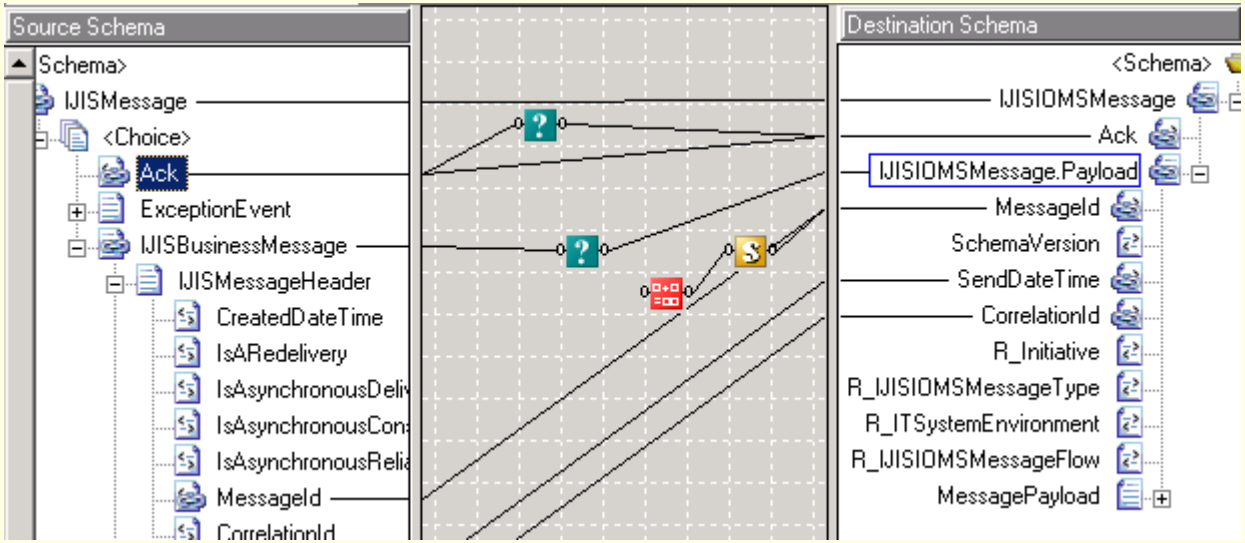
Common Mapping Logic Constructs

- If-Then-Else:



Common Mapping Logic Constructs

- Logical Existence:



Tips & Tricks

- Using labels
- Using SHIFT key for linking all fields within a node
- Moving links across pages
- Replacing functoids
- Using test values (constants)*

** remember to remove test values before deployment!*

Mapper Alternatives: XSLT

- PROS:
  - Direct XSLT is more powerful, fewer limitations than the BizTalk Mapper
  - Improved performance(?)
  - XSLT file can be developed separately and hosted in a BizTalk map
  - Open standard
  - Easier to change/deploy(?)
  - ??
- CONS:
  - Not quite as intuitive
  - Requires “geeky” coding skills
  - Loss of visual map representation
  - ??

Mapper Alternatives: Deserialization

- PROS:
  - Messages can be deserialized using .NET classes
  - Mappings can be assigned using .NET code of choice
  - Exposes functionality of the .NET framework
  - Easier to deploy & maintain *(common skillset)*
  - ??
- CONS:
  - Reduced performance (deserialization overhead)
  - Loss of visual map representation
  - ??

Summary

- The BizTalk Mapper is a powerful GUI tool for generating XSL transformations
- Provides self-contained testing mechanism (within Visual Studio)
- May not always be the best solution for a mapping problem, but adequately covers most scenarios
- Worth the learning curve

Questions?