Pipelines

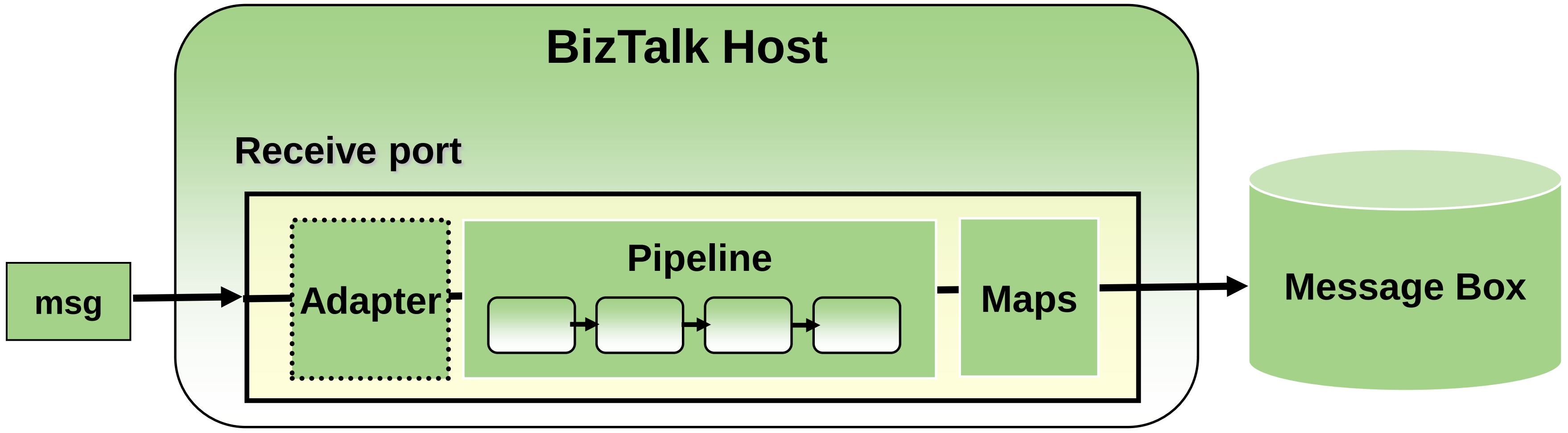
Processing messages entering and exiting BizTalk

Outline

- Pipeline fundamentals
- Pipeline configuration
- Custom pipeline components

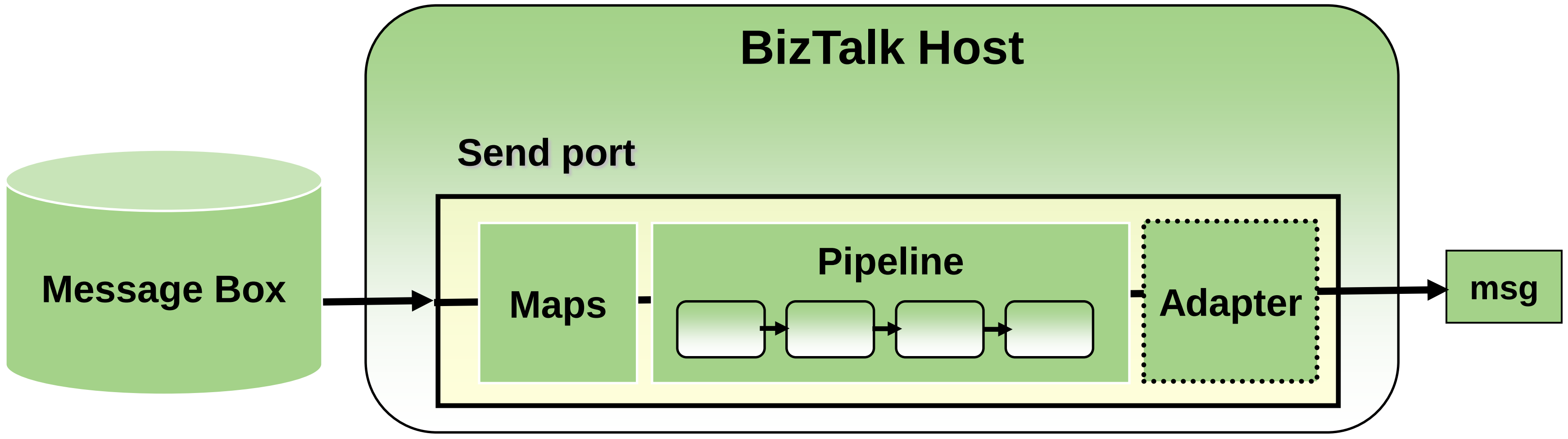
Receiving Messages

- **Adapter receives raw document/data and submits message**
  - Message passes through pipeline
  - Message is processed by a matching map on the port
  - Message type is used to match source of the map



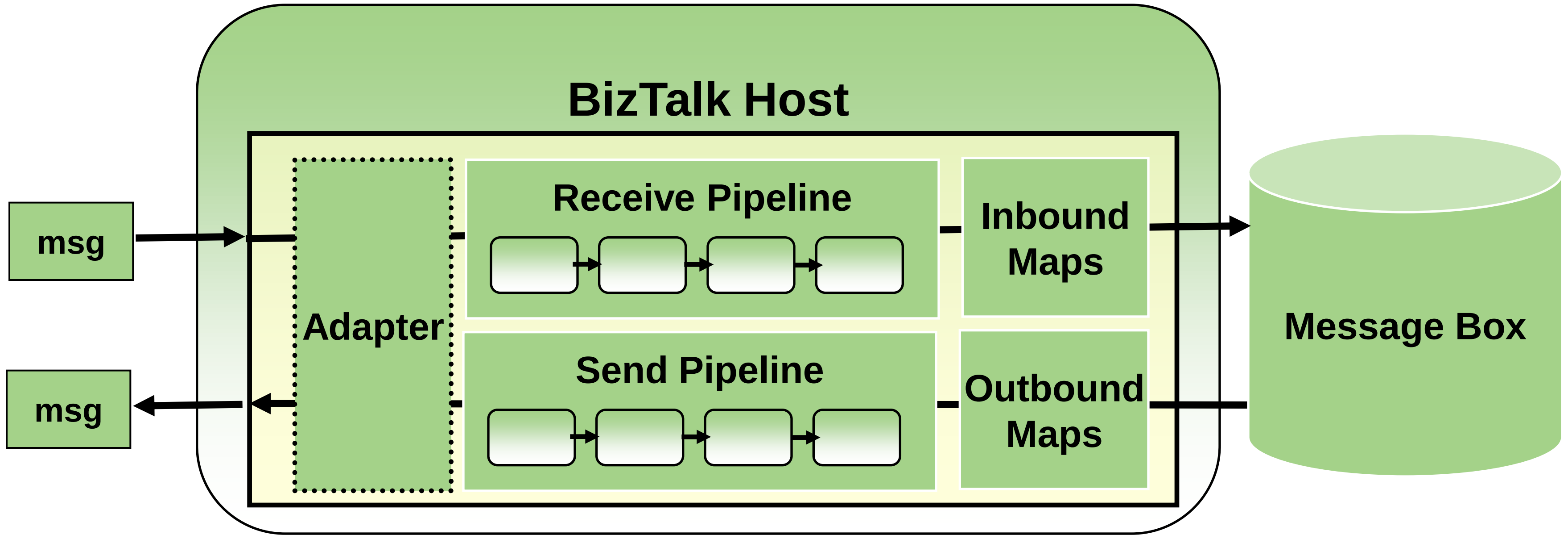
Sending Messages

- **Orchestration or messaging engine sends message**
  - Matching map applied to the data
  - Pipeline executed on the message
  - Adapter given message to transmit



Two-way ports

- Receive = Request / Response
- Send = Solicit / Response



Pipelines

- **Pipelines process messages entering or exiting BizTalk**
  - Provides a model for preparing, massaging messages
  - Messages are streamed through the pipeline for performance
- **Pipelines define a sequence of message processing steps**
  - Organized into well-defined *stages*
  - Each stage may contain zero or more *pipeline components*
  - BizTalk distinguishes between receive and send pipelines

Pipeline stages

Receive pipeline stages

Stage	Description
Decode	Decrypts or decodes the message data
Disassemble	Disassembles an *interchange* into smaller messages (using an *envelope schema*), converts from flat file formats, and parses message content
Validate	Validates message data, generally against a schema
Resolve Party	Identifies the BizTalk Server party associated with some security token in the message/context

Send pipeline stages

Stage	Description
Pre-assemble	Performs and message processing necessary before assembling the message
Assemble	Assembles the message and prepares it to be transmitted by taking such steps as adding envelopes, converting XML to flat files, or other tasks complementary to disassemble
Encode	Encodes or encrypts the message before delivery

Pipelines components

- **A pipeline component defines a processing action**
  - Numerous pipeline components ship with BTS 2009
  - You can write custom pipeline components
- **BizTalk ships several built-in pipeline components**
  - XML assembler/disassembler
  - Flat File assembler/disassembler
  - EDI assembler/disassembler
  - BizTalk Framework assembler/disassembler
  - MIME/SMIME encoding and decoding
  - AS2 encoder and decoder
  - XML validation
  - Party resolution

Default pipelines

- **BizTalk ships several default pipelines for your use**
  - They take advantage of a few built-in pipeline components

Pipeline Name	Description
XMLReceive	Contains the XML Disassembler (builds the message context) and the Party Resolution components
PassThruReceive	Contains no pipeline components
XMLTransmit	Contains the XML Assembler component
PassThruTransmit	Contains no pipeline components
EDISend / EDIReceive	Contain EDI assembler and disassembler components
AS2Send / AS2Receive	Contain AS2 encoder and decoder components
AS2EDISend / AS2EDIReceive	Combine the EDI and AS2 components into a pipeline

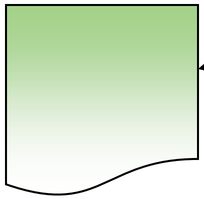
Pipeline configuration

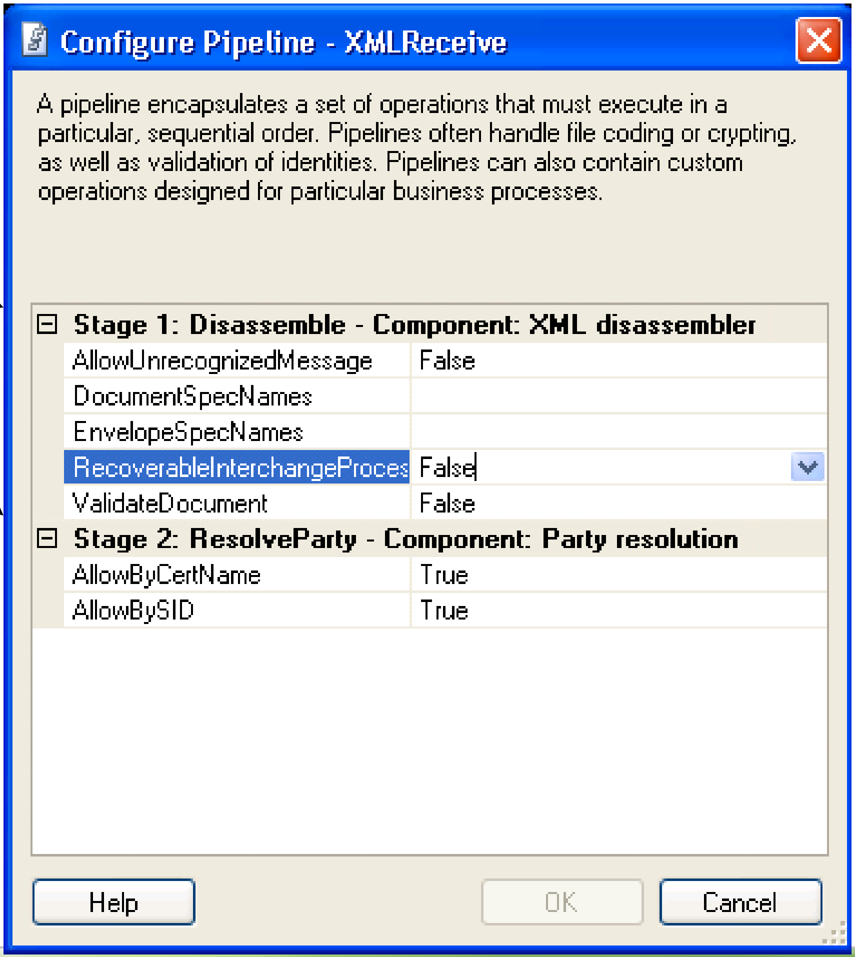
- **Creating a pipeline defines the static pipeline configuration**
  - In many cases you want a template pipeline
  - Same components but different values for different ports
- **Pipeline component properties can be set in the admin tool**
  - Send port and receive location allow for configuring

Per pipeline instance configuration

All stages and components represented

Configuration saved in binding





A pipeline encapsulates a set of operations that must execute in a particular, sequential order. Pipelines often handle file coding or crypting, as well as validation of identities. Pipelines can also contain custom operations designed for particular business processes.

Stage 1: Disassemble - Component: XML disassembler	
AllowUnrecognizedMessage	False
DocumentSpecNames	
EnvelopeSpecNames	
RecoverableInterchangeProces	False
ValidateDocument	False
Stage 2: ResolveParty - Component: Party resolution	
AllowByCertName	True
AllowBySID	True

Custom pipeline components

- **Custom components can be written for any stage of pipelines**
  - Components may replace default implementations
  - May be generic, or very specific to a particular process
  - Developed as .NET or COM components
- **Typical use cases**
  - Customized property promotion
  - Modify message, add or remove parts
  - Alter or inspect streams of data on message parts

Creating custom pipeline components

- **Decorate .NET class with *ComponentCategory* attribute**
  - Indicates this class is a pipeline component
  - Also indicates which stages of execution are appropriate
- **Implement appropriate interfaces**
  - IBaseComponent – properties for Name, Description, Version
  - IComponentUI – designer validation and icon
  - IPersistPropertyBag – support for persisting settings
  - IComponent – Execute method where all the work happens
  - IProbeMessage – for first match stages, indicate a match

Modifying messages in pipelines

- **You must handle messages with care in pipeline components**
  - Remember, messages are generally considered immutable
  - You can promote properties on the original message
- **Otherwise you must clone the message before changing it**
  - You'll need to copy the message, parts, and context
- **BizTalk provides a few helper classes to simplify this process**
  - *PipelineUtil* class provides methods for cloning
  - *PipelineContext* provides other utility methods and properties

Pipeline context

- ***Pipeline context* is passed to the execute method**
  - Provides details about component location in the pipeline
  - Methods to access schemas based on type or name
  - Access to the message factory
  - Factory to create new parts, messages, context and property bags

Using IBaseMessageFactory

- **Interface based model for creating message related items**
  - Critical component when building pipeline components

```
//Get the message factory interface
IBaseMessageFactory factory = pContext.GetMessageFactory();

//Create a new message and clone the context
IBaseMessage pOutMsg = factory.CreateMessage();
pOutMsg.Context = PipelineUtil.CloneMessageContext(pInMsg.Context);

//Create message part and set body and properties
IBaseMessagePart body = factory.CreateMessagePart();
body.Data = pInMsg.BodyPart.GetOriginalDataStream();
body.PartProperties = pInMsg.BodyPart.PartProperties;

//Add part to the message
pOutMsg.AddPart(pInMsg.BodyPartName, body, true);
```

Message context properties

- **Use *IBaseMessageContext* to handle context properties**
  - Always need to reference the qualified name of the property
  - Qualified name is available on the .NET type for property schemas
  - Distinguished properties are written to the context
  - Promoted properties are promoted to the context
  - Both can be read from the context

```
//write a distinguished field to the context
msg.Context.Write("theDistinguishedProperty",
         "http://schemas.microsoft.com/BizTalk/2003/btsDistinguishedFields",
         "theDistinguishedValue");

//promote a property to the context
msg.Context.Promote(bam.Name.Name, bam.Name.Namespace, bamGuid.ToString());

//read a promoted property from the context
SOAP.UserDefined user = new SOAP.UserDefined();
string val = (string)msg.Context.Read(user.Name.Name, user.Name.Namespace);
```

Deploy pipeline components

- **Pipeline components can be deployed in two locations**
  - Directory: [BTSINSTALL_DIR]\Pipeline Components
  - Global Assembly Cache (GAC)
- **Must deploy to the GAC if using pipelines in orchestration**
  - Recommended to deploy all custom components to GAC

Summary

- Pipelines are for processing messages entering or exiting
- Custom pipeline components are just .NET classes