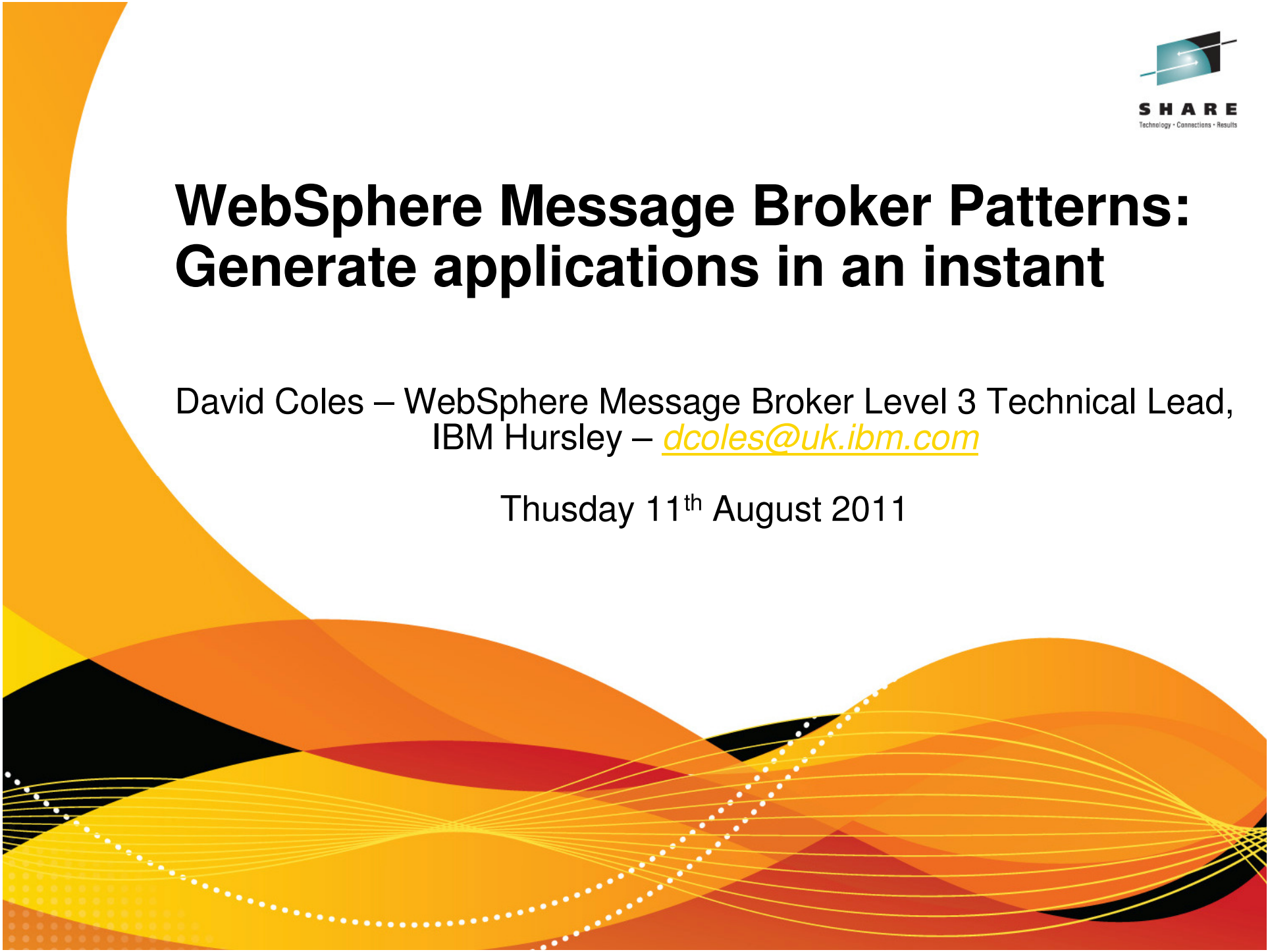
# WebSphere Message Broker Patterns: Generate applications in an instant

David Coles – WebSphere Message Broker Level 3 Technical Lead,
IBM Hursley – dcoles@uk.ibm.com

Thusday 11th August 2011



# Agenda

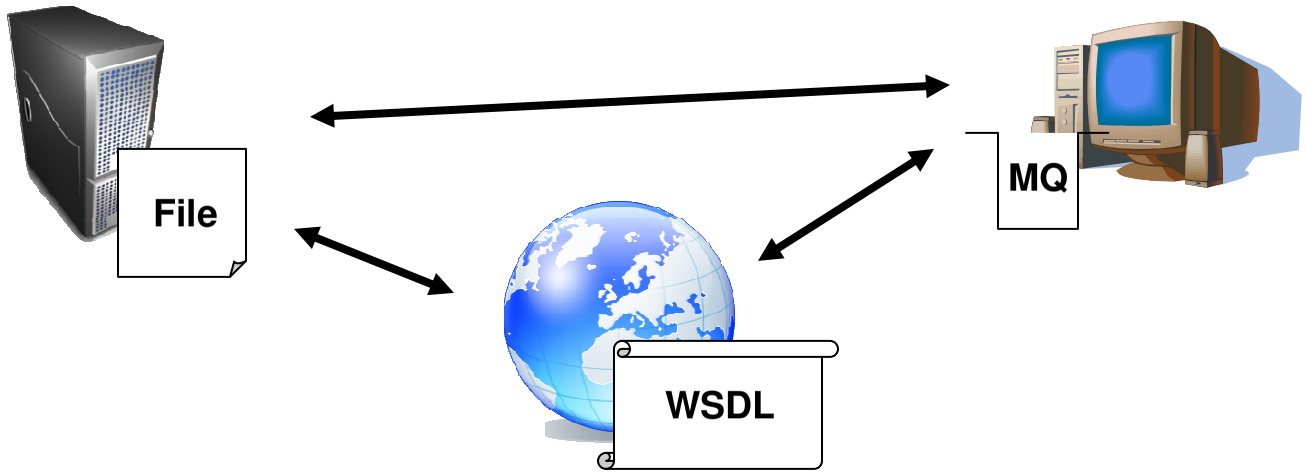

- Overview
- Patterns
  - Broker Navigator
  - Patterns Explorer
  - Pattern Parameters
  - Generation
  - Deployment
  - Built-in Patterns
- Pattern Authoring
  - Principles and Workflow
  - Design Walk-Through
  - Expressions, Enablement & Enumerated Types
- Pattern Refinement
  - Java
  - PHP
  - Debugging
- Pattern Communities

# Overview

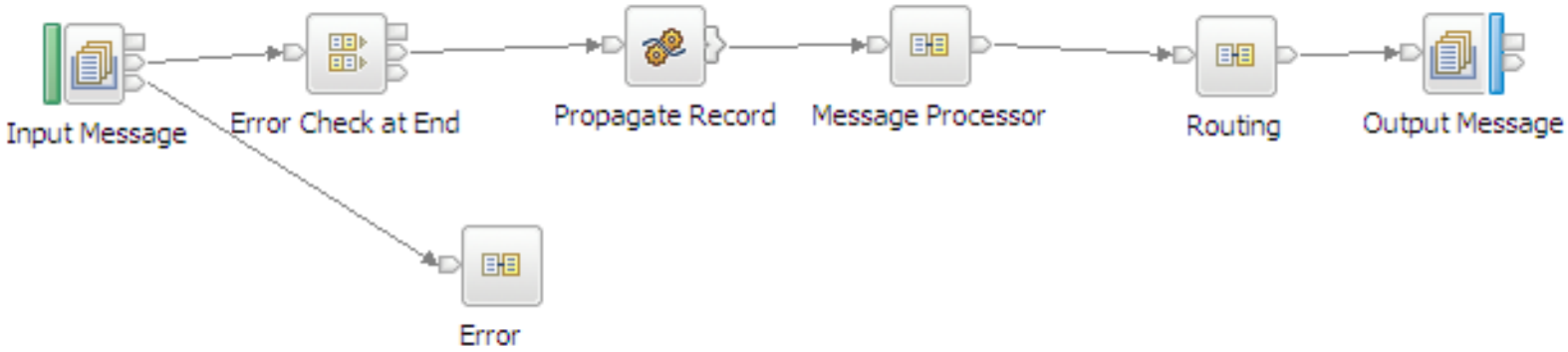
- Universal Connectivity
  - Simplify application connectivity to provide a flexible and dynamic infrastructure
  - Routes and transforms messages from anywhere, to anywhere
  - Supports a wide range of protocols
    - MQ, JMS 1.1, HTTP(S), Web Services (SOAP, REST), file, ERP (SAP, SEBL…), TCP/IP
  - Supports a broad range of data formats
    - Binary (C/COBOL), XML, CSV, Industry (SWIFT, EDI, HL7…), IDOCs, user defined
- Interactions and Operations
  - Route, filter, transform, enrich, monitor, distribute, decompose, correlate, detect…
- Simple programming
  - Patterns based for top-down, parameterized connectivity of common use cases
    - Web Service façades, message oriented processing, queue to file…
  - Construction based for bottom-up assembly of bespoke connectivity logic
    - Message flows to describe application connectivity comprising…
    - Message nodes which encapsulate required integration logic which operate on…
    - Message tree which describes the data in a format independent manner
    - Transformation options include graphical mapping, PHP, Java, ESQL, XSL and WTX
- Operational Management and Performance
  - Extensive administration and systems management facilities for developed solutions
  - Wide range of operating system and hardware platforms supported
  - Offers performance of traditional transaction processing environments

# The Challenge

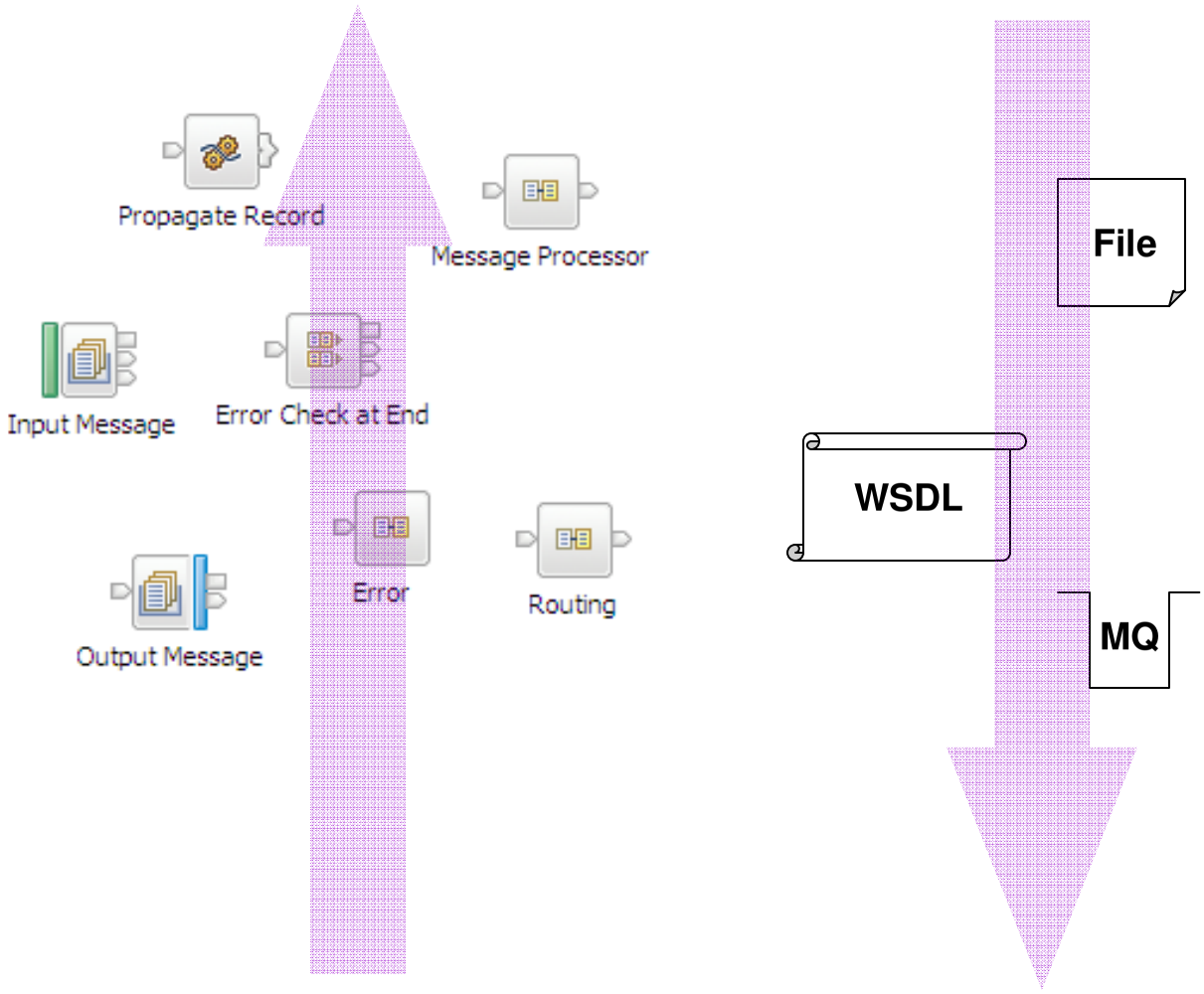
From:




To:



# Top-down vs Bottom-up Development



# Patterns for Simplified Development

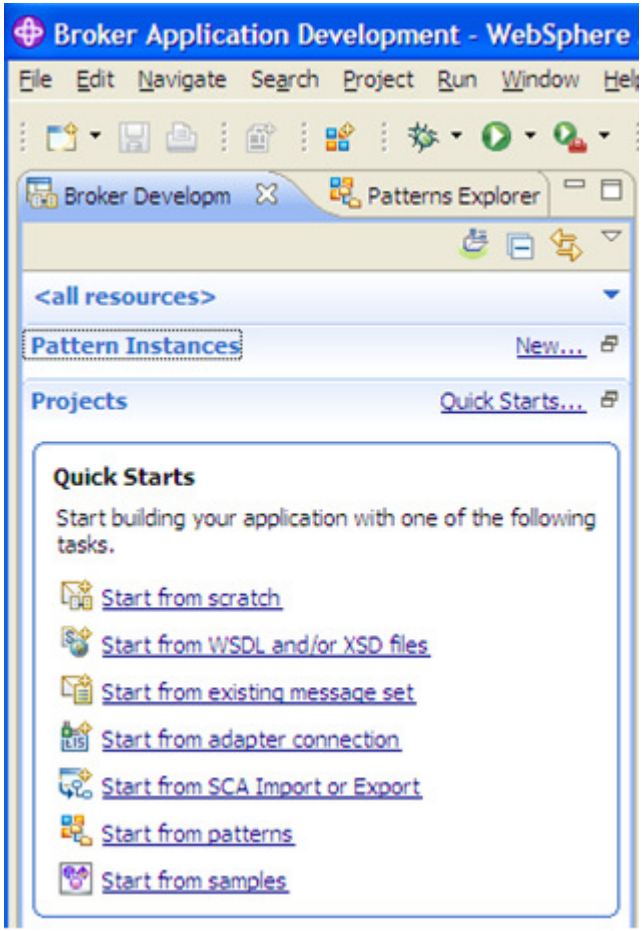

- Creates top-down, parameterized connectivity solutions
  - Web Service façades, message oriented processing, queue-to-file
- Reduces common problems in flow development
- Communicates best practices to the Broker community
- Complements existing bottom-up construction for bespoke connectivity
- Reduces time-to-value for solution development
- Patterns are a first class citizen in Message Broker
  - Patterns have bubbled right to the top in the navigator view!


SHARE
Technology · Connections · Results

# Patterns

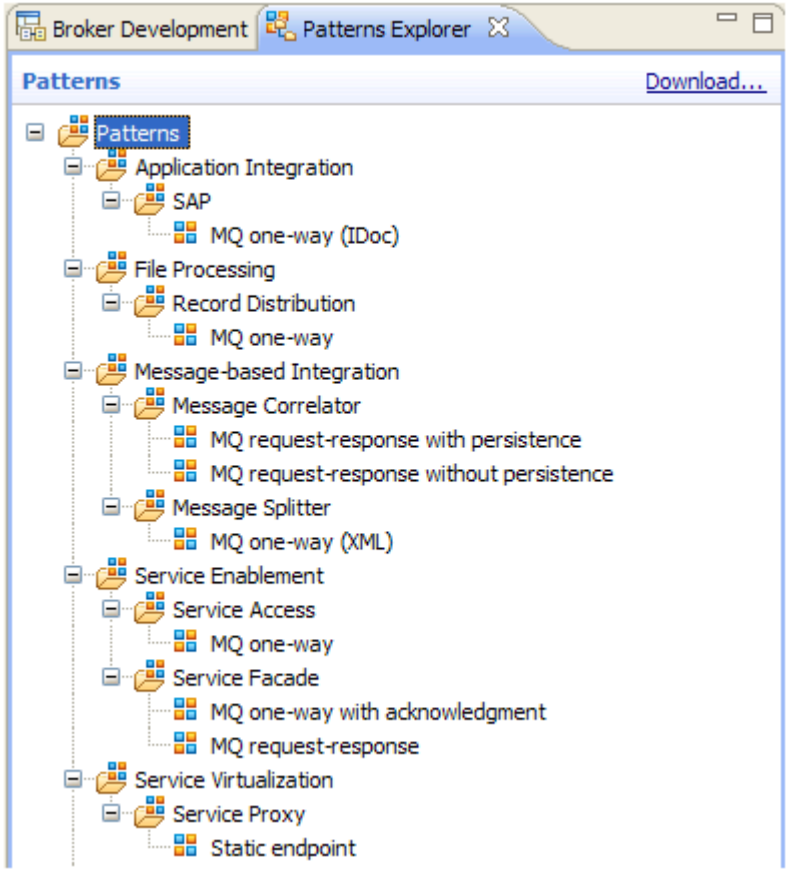

SHARE
in Orlando
2011

# Message Broker Navigator




- User is directed towards a new entry point for development
- First class development mechanism
- Augments other starting points:
  - WSDL, SCA, Adapter
- Does not replace the existing bottom up development approach
- Still completely valid to start from flows, message sets etc!

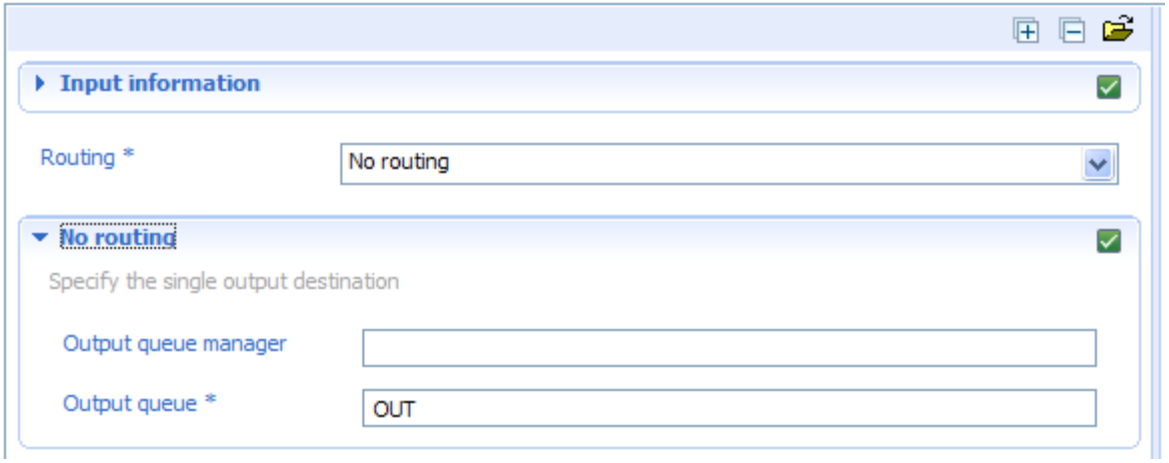
# Patterns Explorer




- Patterns Explorer
  - Pre-supplied IBM patterns
- Pattern categories
  - Groups similar solutions together
  - Message-based integration, service enablement, service virtualization…
- Extensive help is available through a pattern specification
  - Provides selection and implementation guidance for the patterns

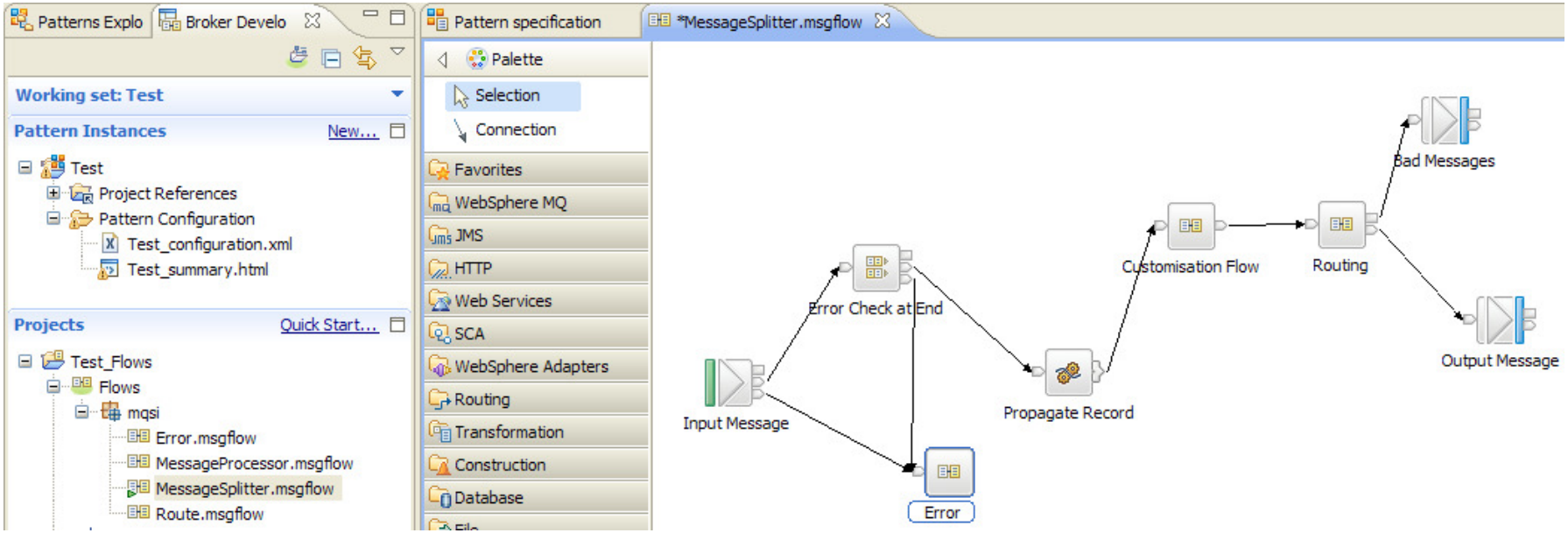
# Pattern Parameters

- Customize a pattern using pattern parameters
  - Instance name identifies pattern; duplicate names will optionally be overwritten
  - Parameters are logically grouped into sections
  - Mandatory parameters are indicated via *, missing parameters are indicated via x
  - Fields are watermarked and pre-populated, for example: available message sets
  - Detailed help is available for each pattern parameter
  - Click the ***Generate*** button to create the generated artefacts: message flows, scripts…



# Generation

- A working set is created, it includes only the current pattern
- The ***Projects*** view contains the generated projects
- The ***Patterns Instances*** view contains the pattern instance projects
  - Includes the configuration XML and summary page
- Patterns do not attempt to provide life cycle management!

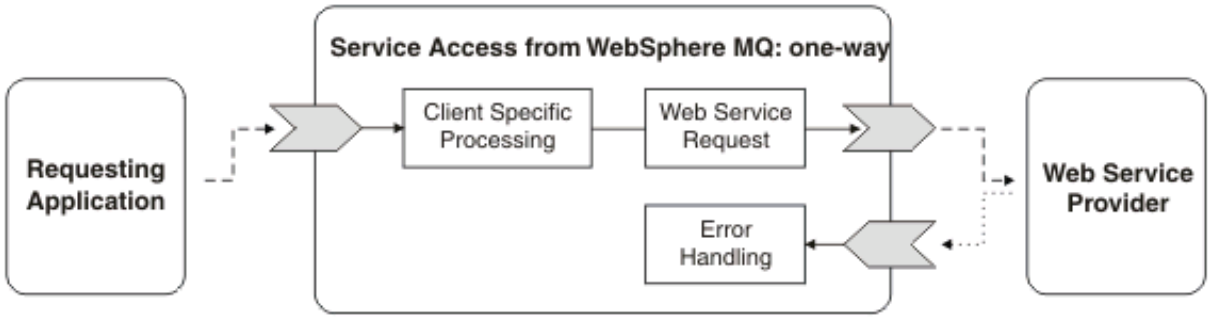


# Deployment

- Nothing changes from regular build and deploy activities
  - Build BAR file as usual from generated assets
  - Deploy as usual, through MBX or the toolkit
- Pattern instance may create additional runtime controls
  - For example, a UDP to control error logging/trace
  - These can be customized as normal

# Built-In Patterns

- Message Broker provides a core set of built-in patterns
- These implement a variety of common scenarios
  - Web service front end to a MQ based application
  - Processing data stored in a file and routing to one or more queues
  - Adding a proxy in front of a web service provider
  - Processing data from an SAP system and routing to MQ
  - Shredding messages and routing to one or more queues
- Patterns are selected based on client feedback and field experience
- This core set of patterns continues to grow with each release
  - MQ to web service pattern added in Message Broker v7 FP1




# Pattern Authoring

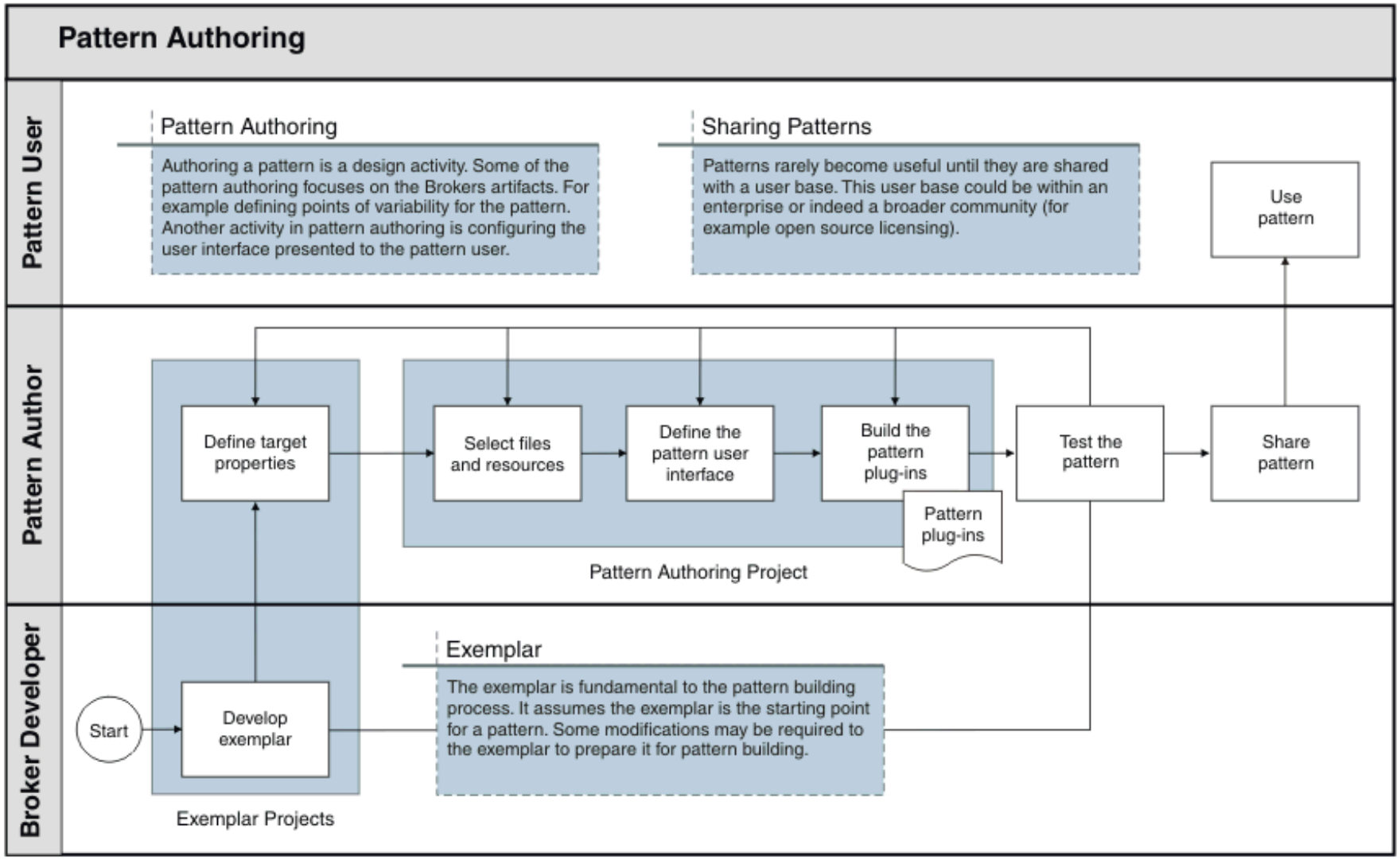
# Principles (Part I)

- Patterns and pattern authoring are first class concepts
- Pattern authoring is a design activity
  - It may be long lived
  - It is often not sequential
- Using patterns is a top-down activity driven by a requirement, but:
  - Authoring an exemplar is (typically) a bottom-up activity
  - So pattern authoring must bridge these two different approaches
- Patterns have their own development cycle
- We always start with a working exemplar - one or more Broker projects
- Creating the exemplar is part of the pattern authoring process

# Principles (Part II)

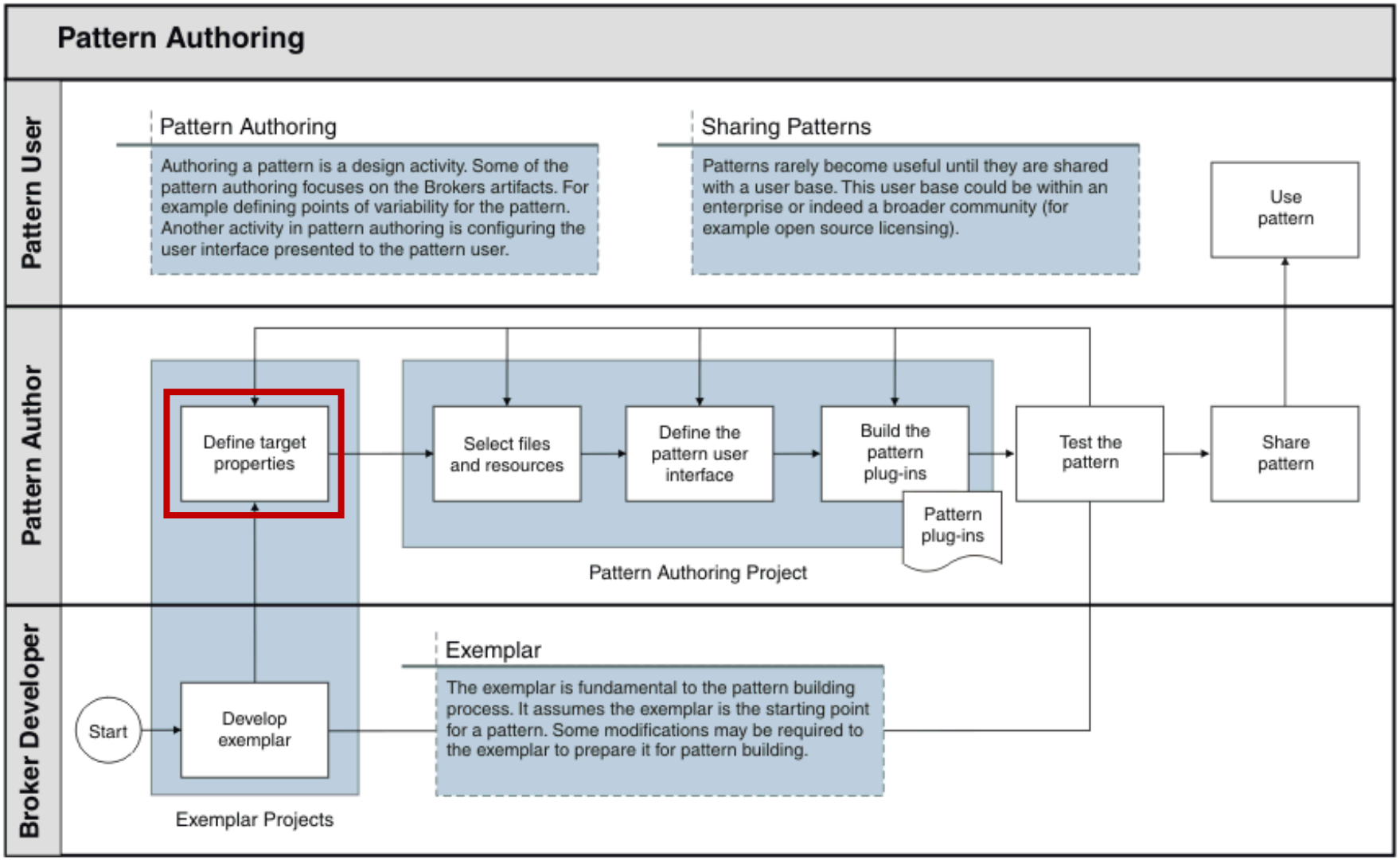
- The Broker implementation of pattern authoring is a Toolkit experience
- Pattern authoring is a managed user experience - a user should not need to understand Broker internals, or other implementation technologies
- A pattern encapsulates one or more exemplar projects and a set of configuration that:
  - Looks down to the exemplar projects to select projects, files and properties
  - Looks up to the user interface that will be presented to the pattern user
  - Allows pattern authors to extend their pattern with PHP and Java code
  - Describes a pattern and its categories through an HTML specification
- Provide guidance so that a pattern author can create exemplars ready for pattern authoring
- Pattern authoring creates patterns whose value multiplies as they are shared and used by a community of developers!

# Workflow

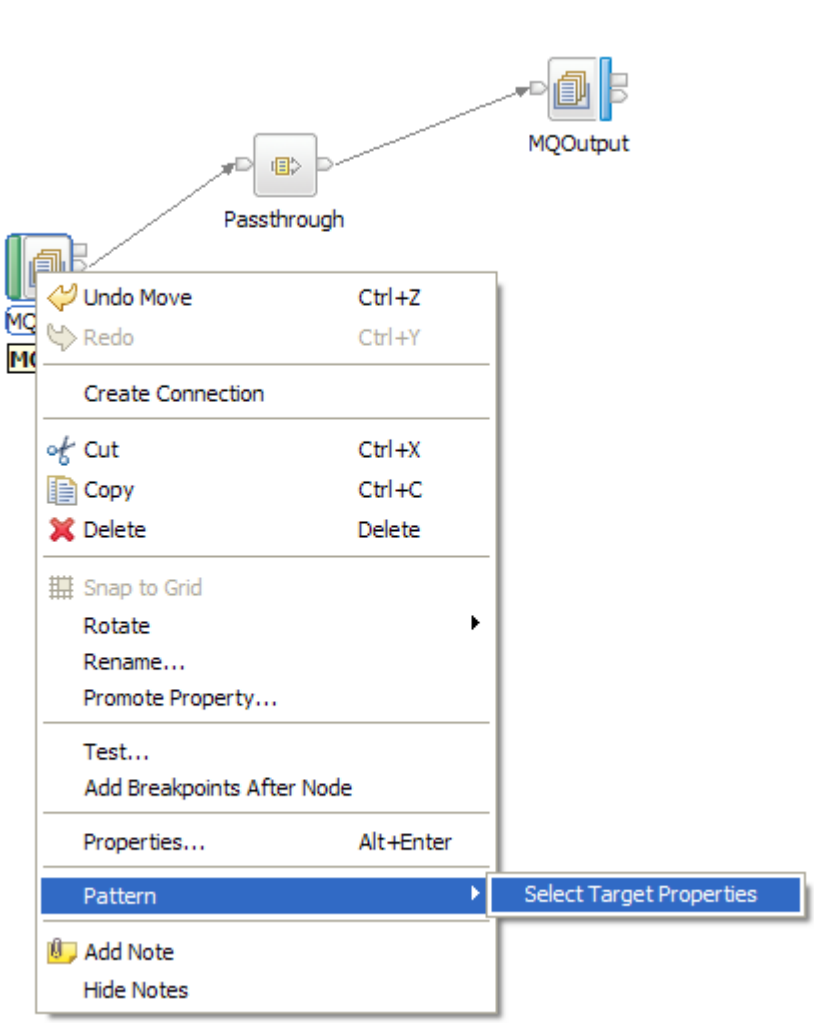

Pattern Authoring
Pattern User
Pattern Author
Broker Developer
Pattern Authoring
Authoring a pattern is a design activity. Some of the pattern authoring focuses on the Brokers artifacts. For example defining points of variability for the pattern. Another activity in pattern authoring is configuring the user interface presented to the pattern user.
Sharing Patterns
Patterns rarely become useful until they are shared with a user base. This user base could be within an enterprise or indeed a broader community (for example open source licensing).
Use pattern
Define target properties
Select files and resources
Define the pattern user interface
Build the pattern plug-ins
Pattern plug-ins
Test the pattern
Share pattern
Pattern Authoring Project
Start
Develop exemplar
Exemplar
The exemplar is fundamental to the pattern building process. It assumes the exemplar is the starting point for a pattern. Some modifications may be required to the exemplar to prepare it for pattern building.
Exemplar Projects

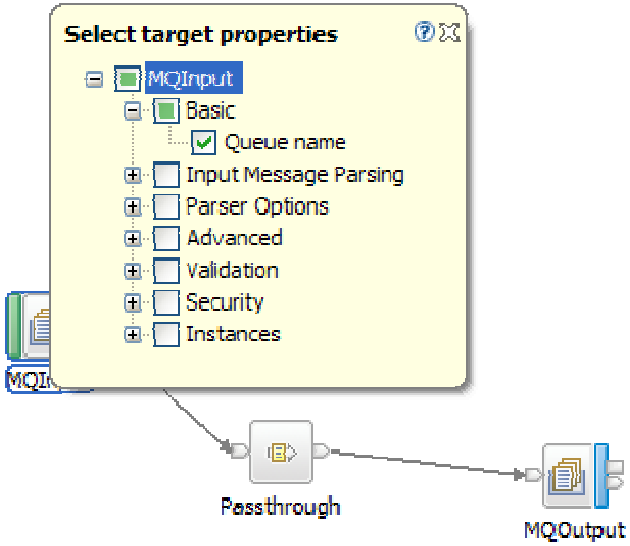

# Design Walk Through

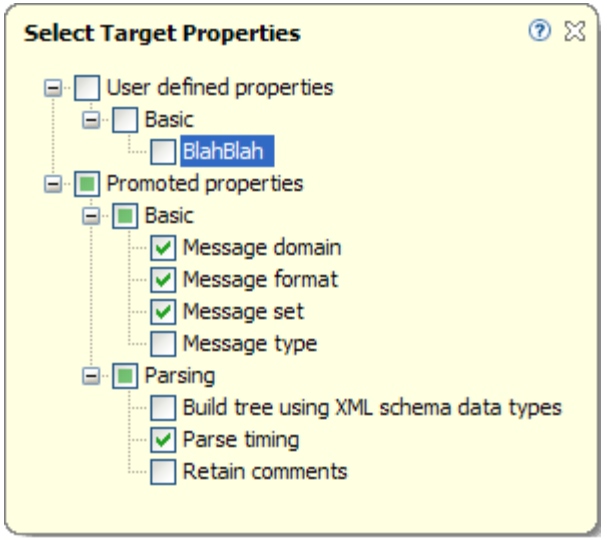
# Define the Target Properties



# Define the Target Properties



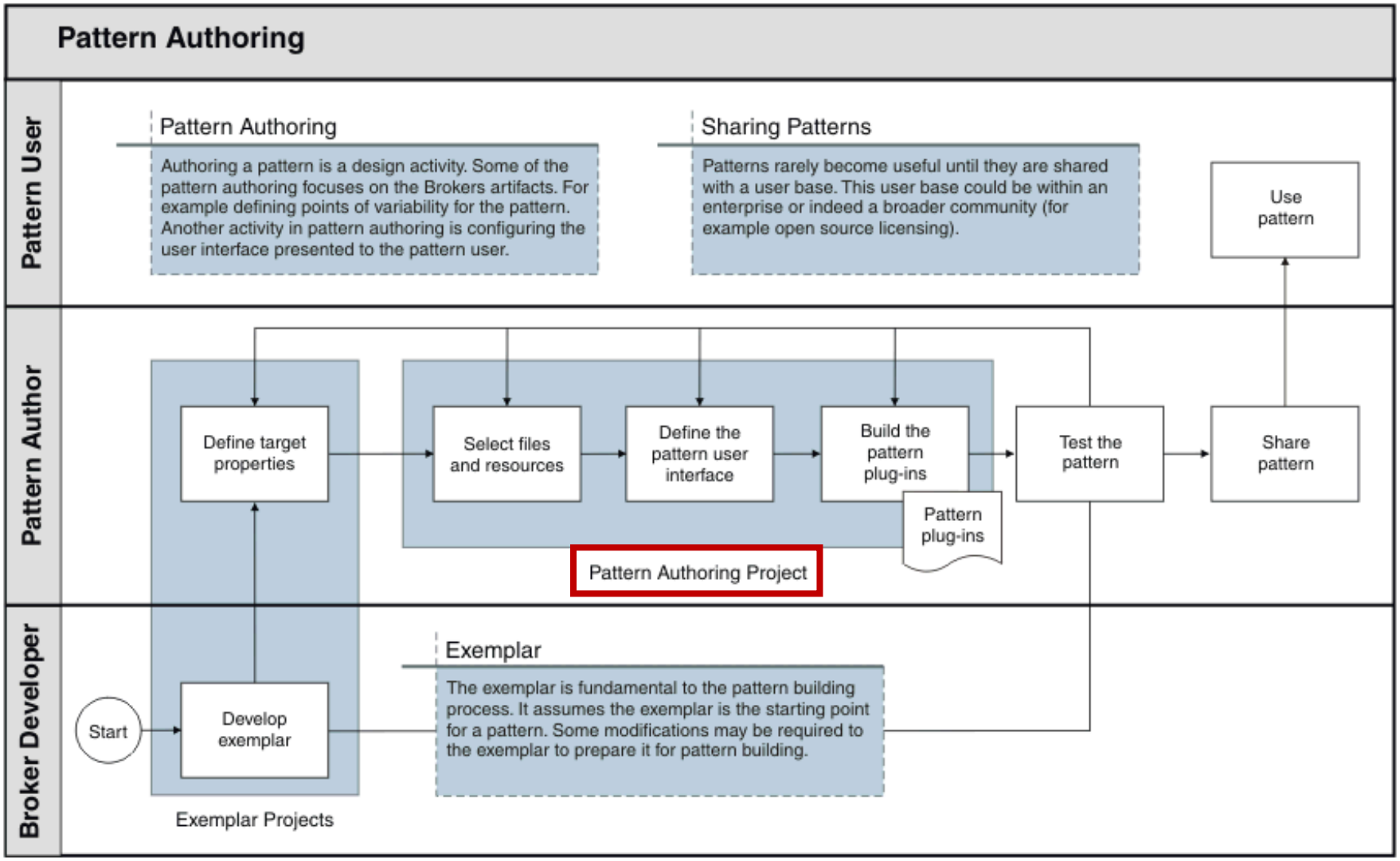



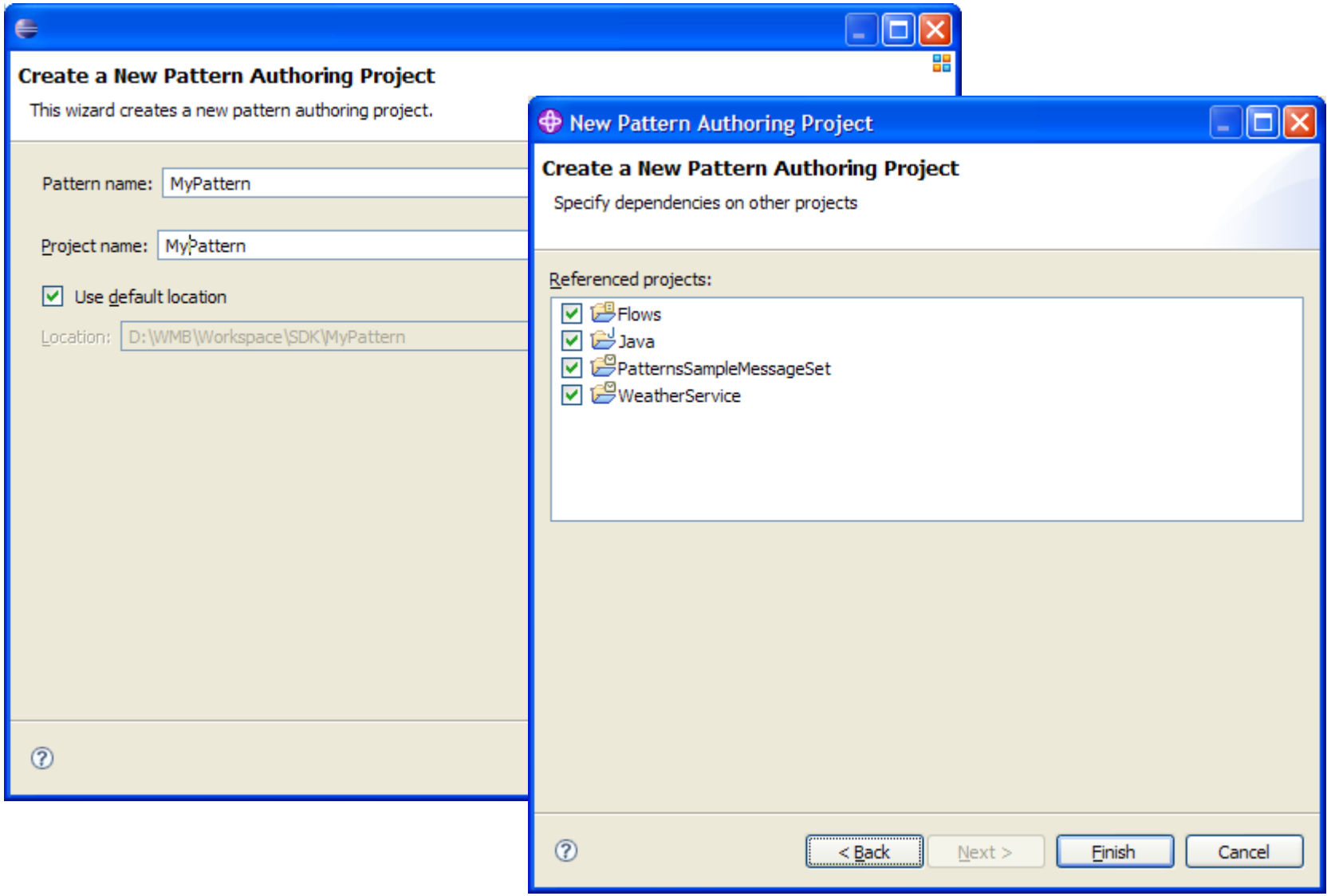
# Design Points

- Patterns are created in the Broker Toolkit
- Patterns are designed in a Pattern Authoring project and editor
- Variability is expressed in the Toolkit editors
  - The first release supports property variability in the Flow Editor
  - Property variability is based on flow and User Defined Properties (UDPs)
  - Adding variability in the editors also supports a use case where exemplar projects are used in multiple patterns

# Create A Pattern Authoring Project



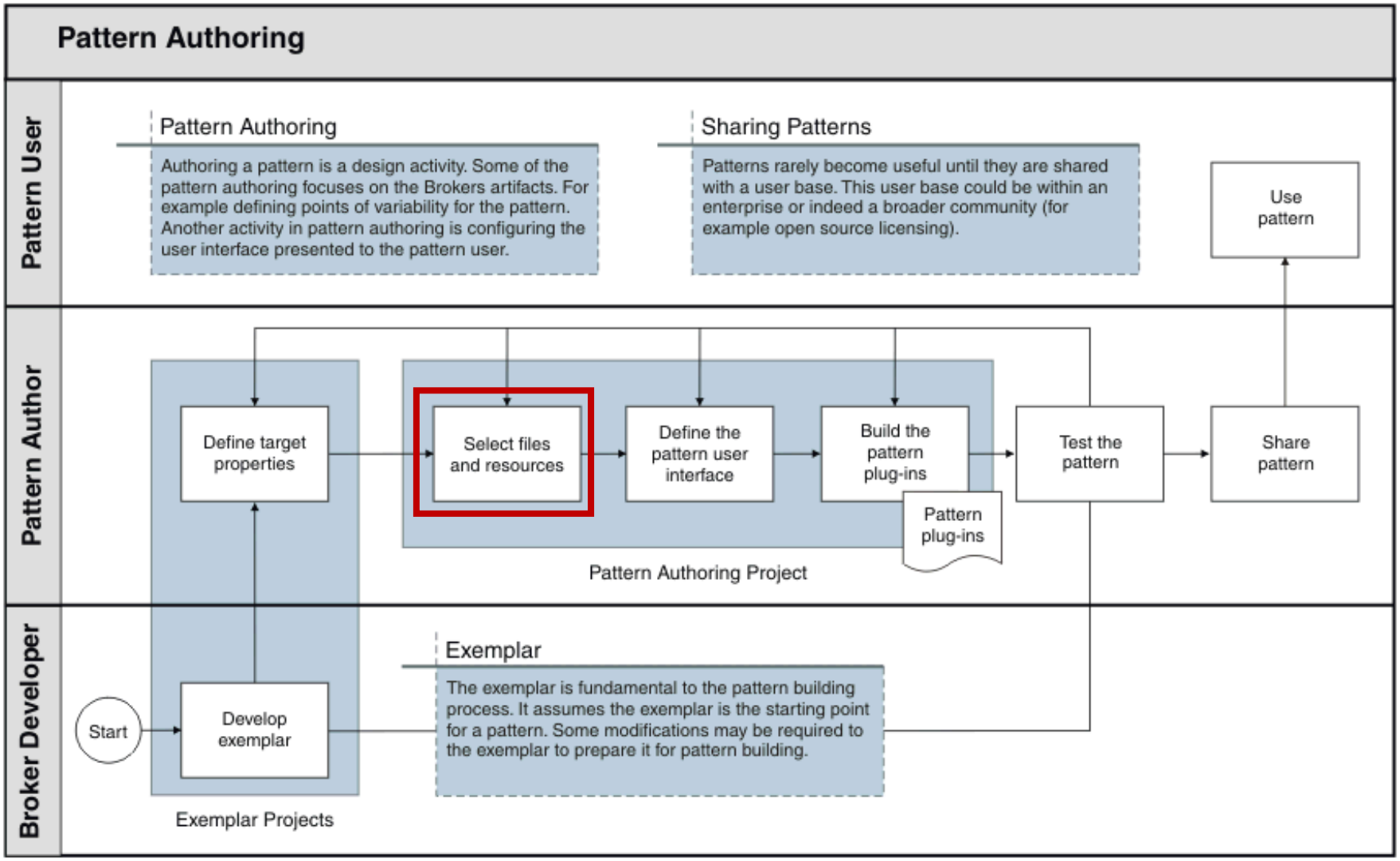
# Create A Pattern Authoring Project



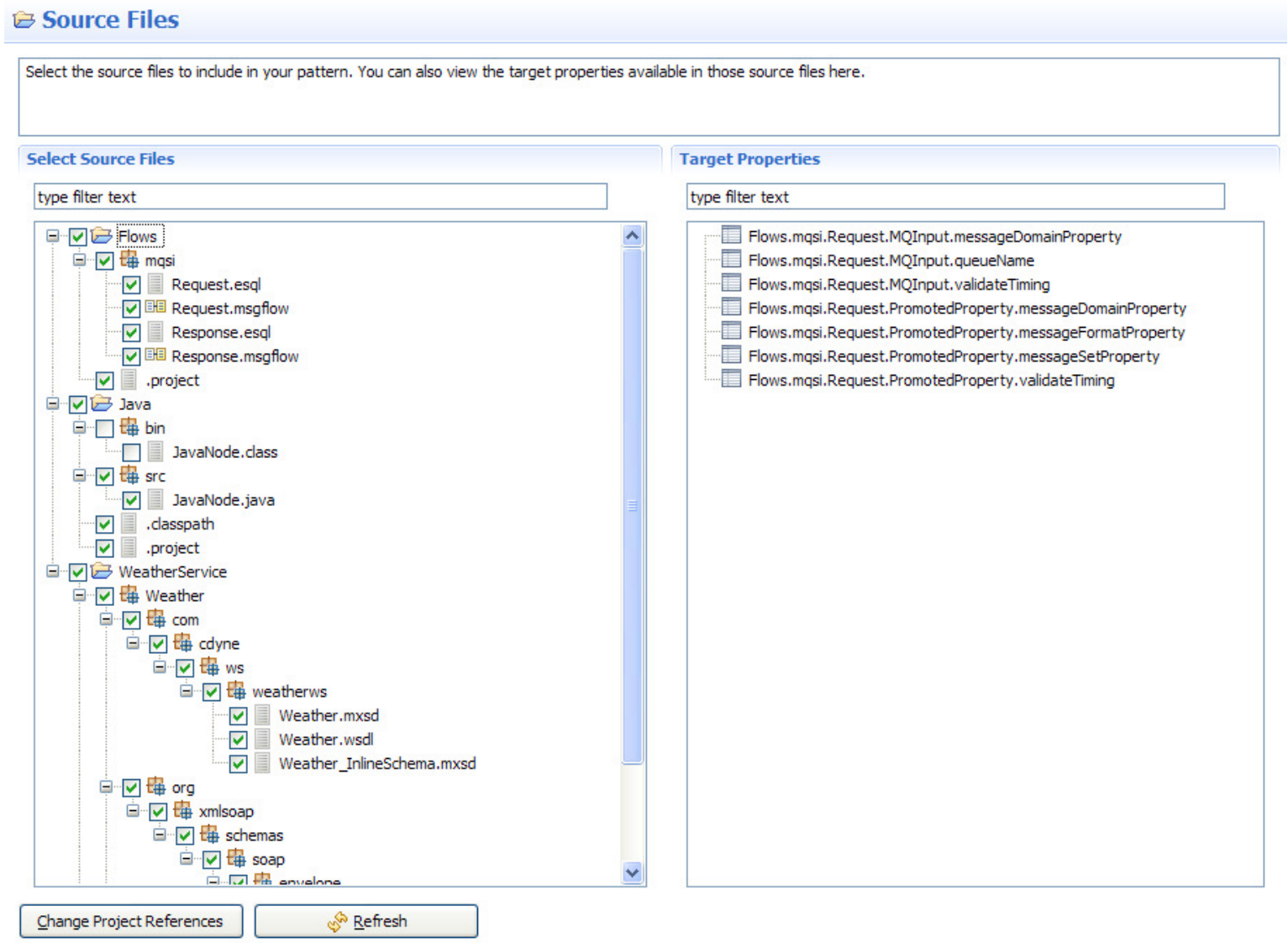
# Design Points

- A pattern can generate one or more message flow projects
- A pattern can contain references to projects which the end user is expected to have in their workspace
- A pattern generates projects that match the projects in the exemplars
  - Projects are named following pattern instance naming conventions. For example, if the exemplar project name is `invoicing` and the pattern instance name is `foo`, then the generated project will be called `foo_invoicing`. This simple rule favours convention over configuration. The pattern author can name their exemplar projects accordingly.
- The pattern author selects their exemplar projects when a new Pattern Authoring project is created
  - The Pattern Authoring project maintains references to the exemplar projects
    - These references are standard Eclipse project dependencies
  - The exemplar project files are copied into the pattern plug-in when the pattern is built

# Select The Files and Resources



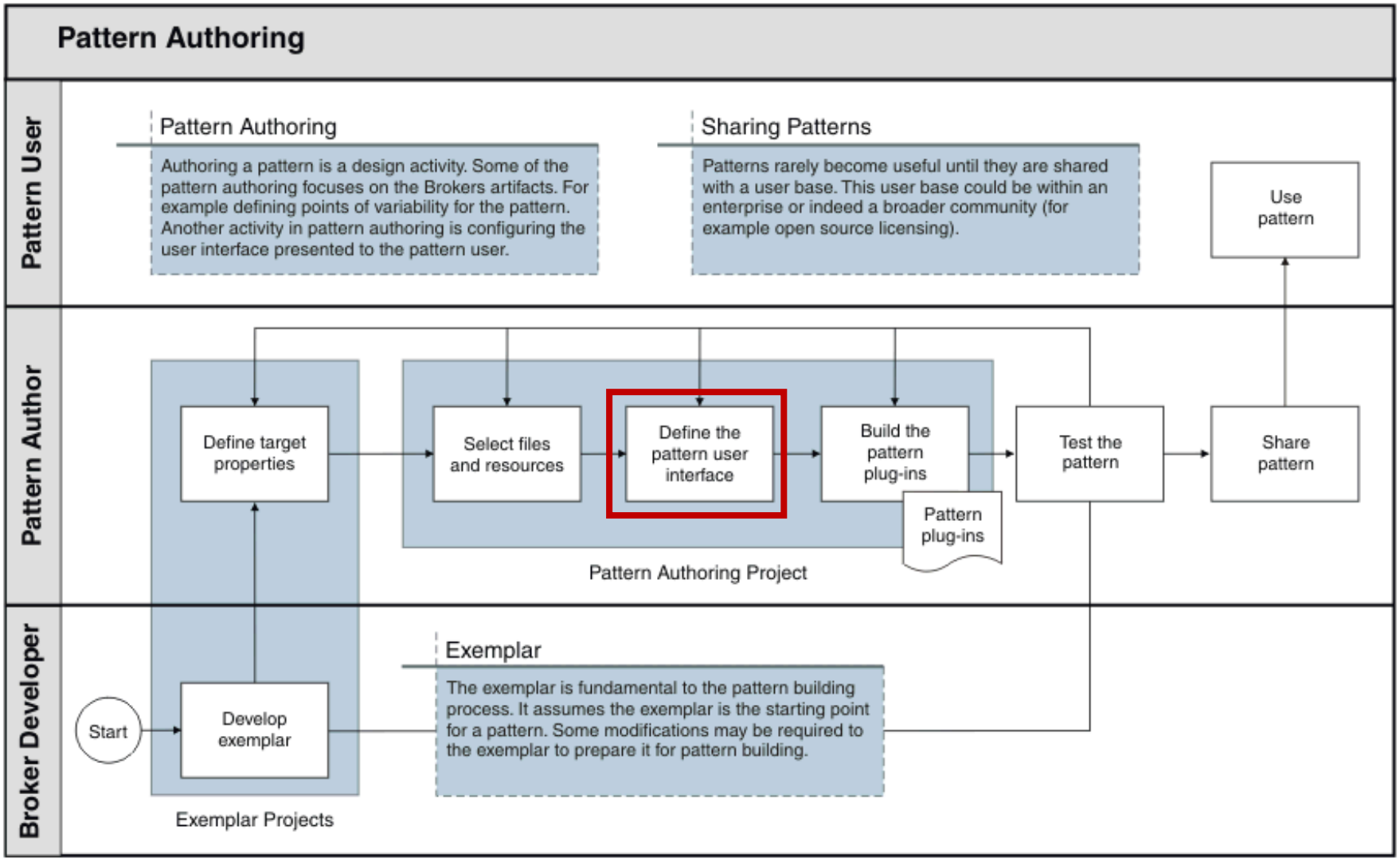
# Select The Files and Resources



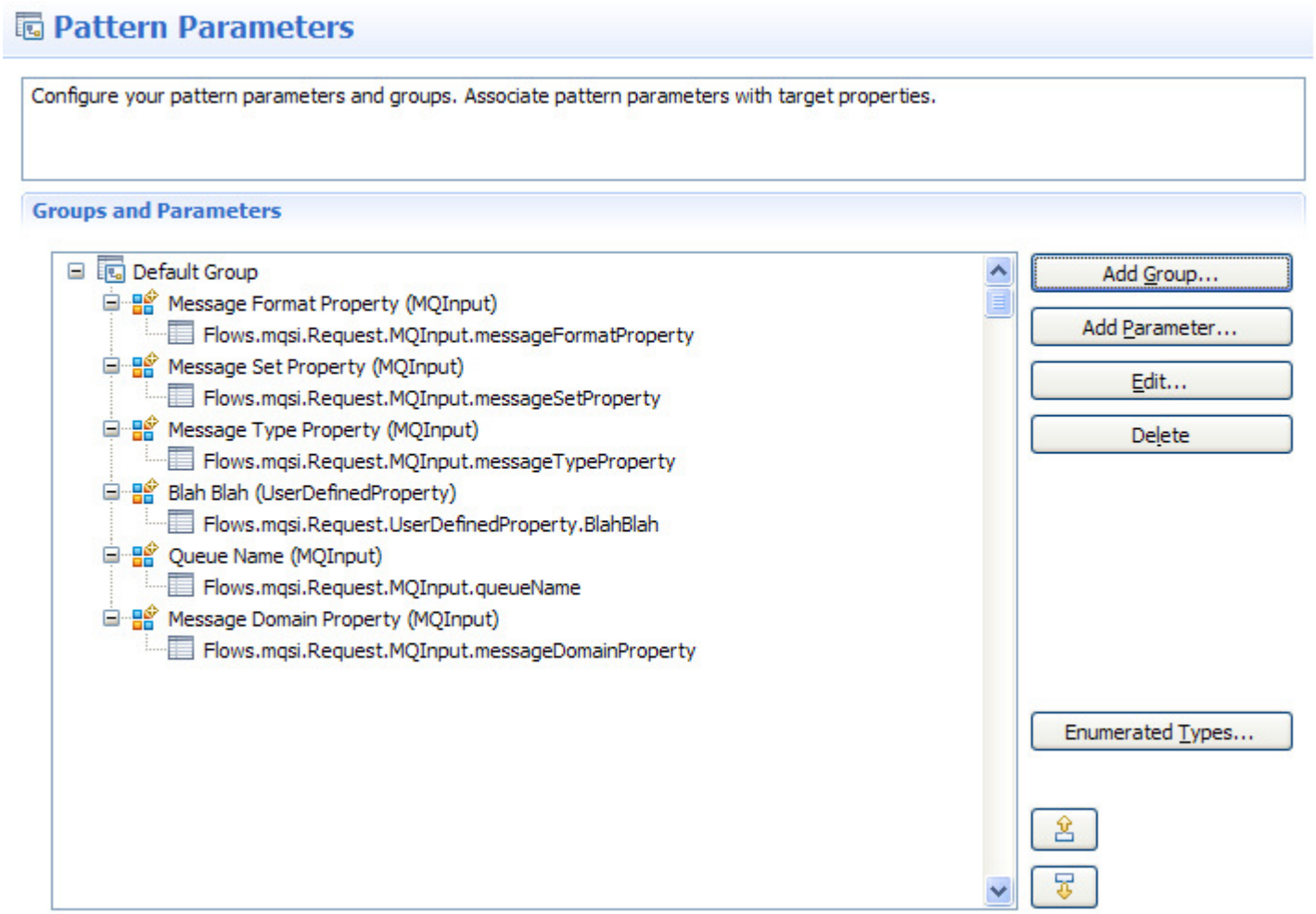
# Design Points

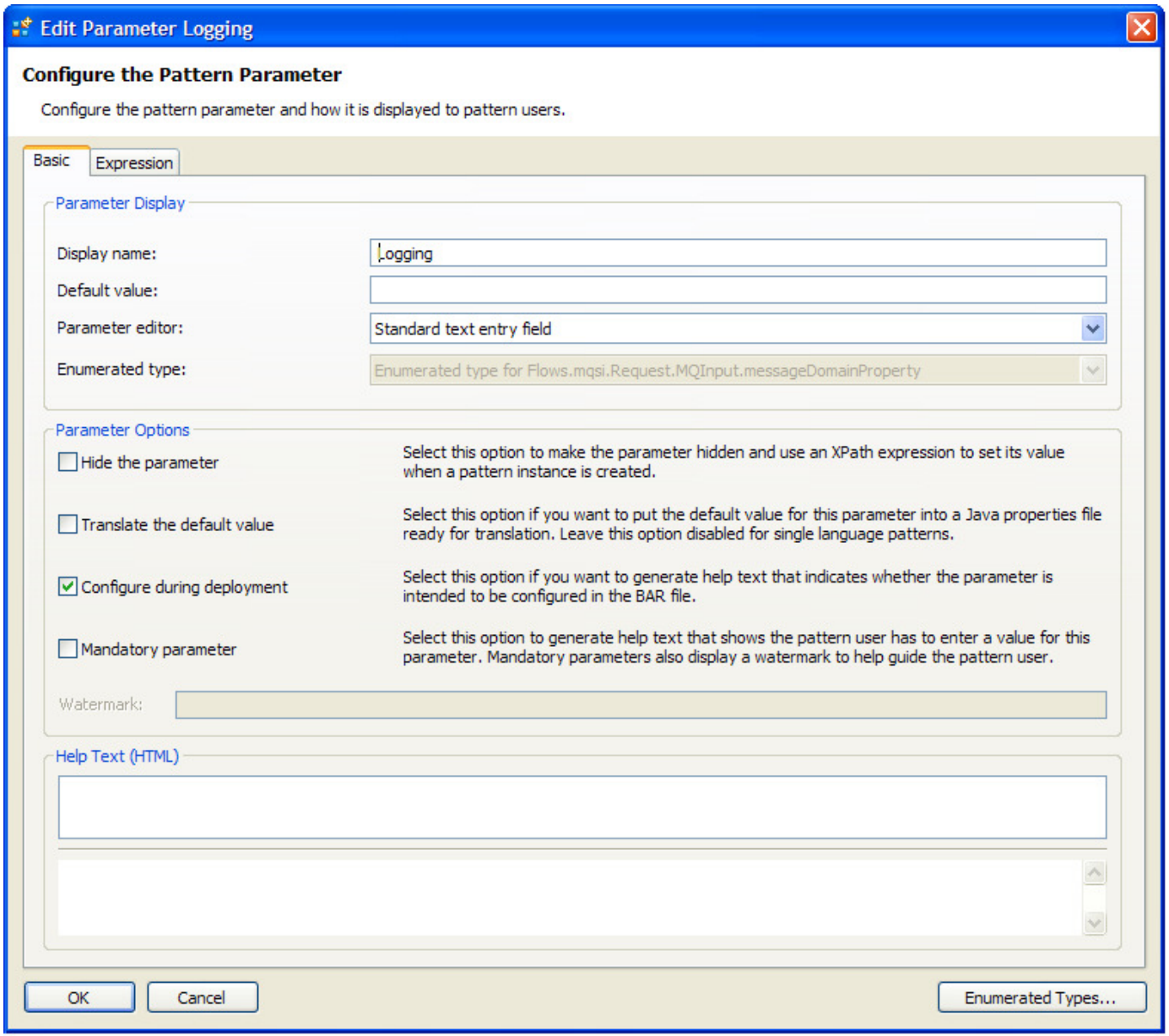
- A pattern author chooses the files to include from their exemplar projects
  - This supports a use case where a pattern author has a library project that contain many re-usable assets not all of which are applicable to any given pattern.
- Choosing a file implicitly selects all target properties in that file
  - By default, all files in an exemplar project are selected
- Target properties can be added and removed in the exemplar projects
  - The Pattern Authoring project can be refreshed to pick these up
  - Double clicking a file opens the editor
  - Authoring a pattern is a non sequential design activity!
- Depending on the project type, some files and directories may need to be excluded (such as the `bin` directory in a Java project)

# Define The User Interface

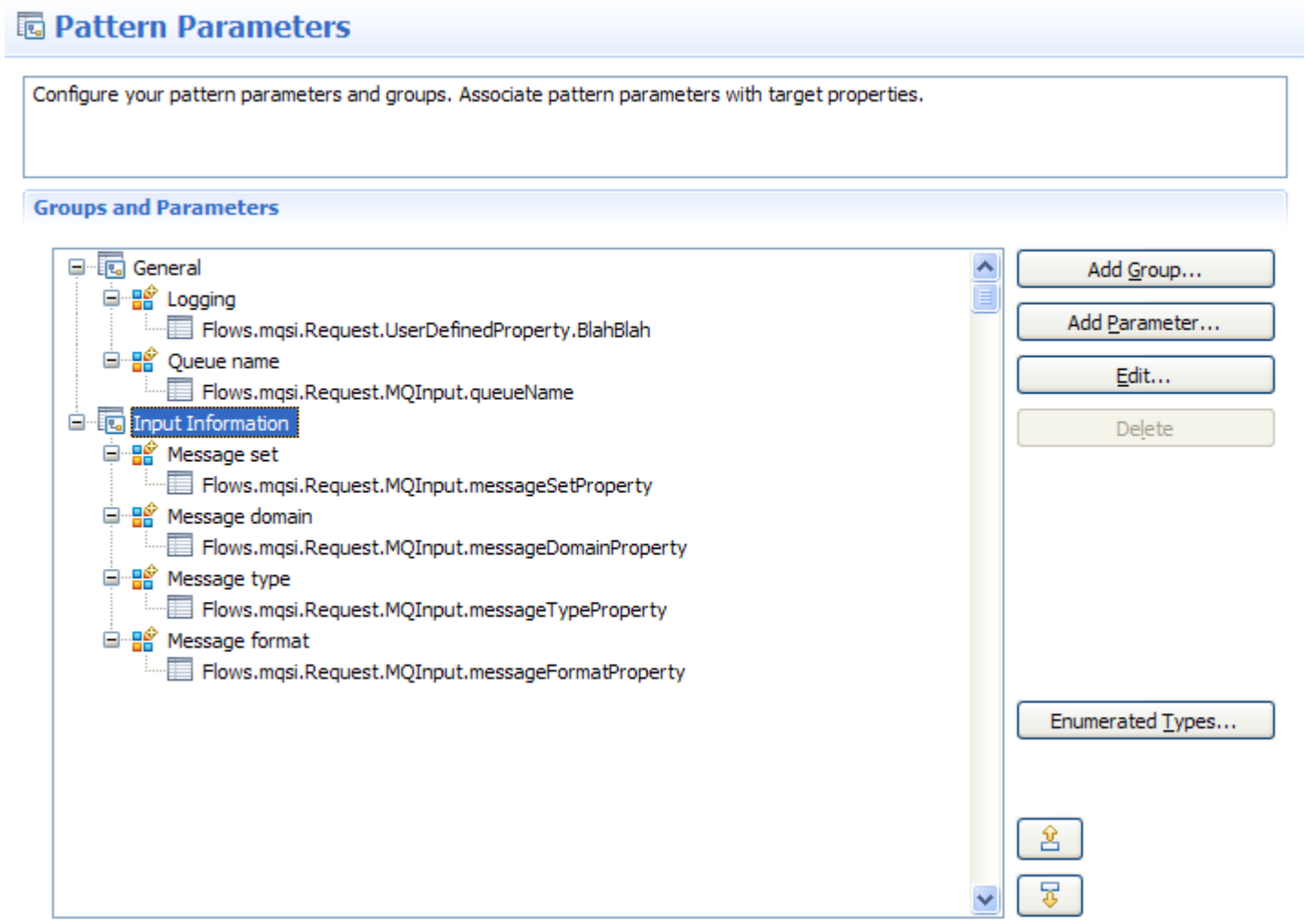


# Configure Pattern Parameters (Before)




Edit Parameter Logging
Configure the Pattern Parameter
Configure the pattern parameter and how it is displayed to pattern users.
Basic
Expression
Parameter Display
Display name:
Logging
Default value:
Parameter editor:
Standard text entry field
Enumerated type:
Enumerated type for Flows.mqsi.Request.MQInput.messageDomainProperty
Parameter Options
Hide the parameter
Select this option to make the parameter hidden and use an XPath expression to set its value when a pattern instance is created.
Translate the default value
Select this option if you want to put the default value for this parameter into a Java properties file ready for translation. Leave this option disabled for single language patterns.
Configure during deployment
Select this option if you want to generate help text that indicates whether the parameter is intended to be configured in the BAR file.
Mandatory parameter
Select this option to generate help text that shows the pattern user has to enter a value for this parameter. Mandatory parameters also display a watermark to help guide the pattern user.
Watermark:
Help Text (HTML)
OK
Cancel
Enumerated Types...

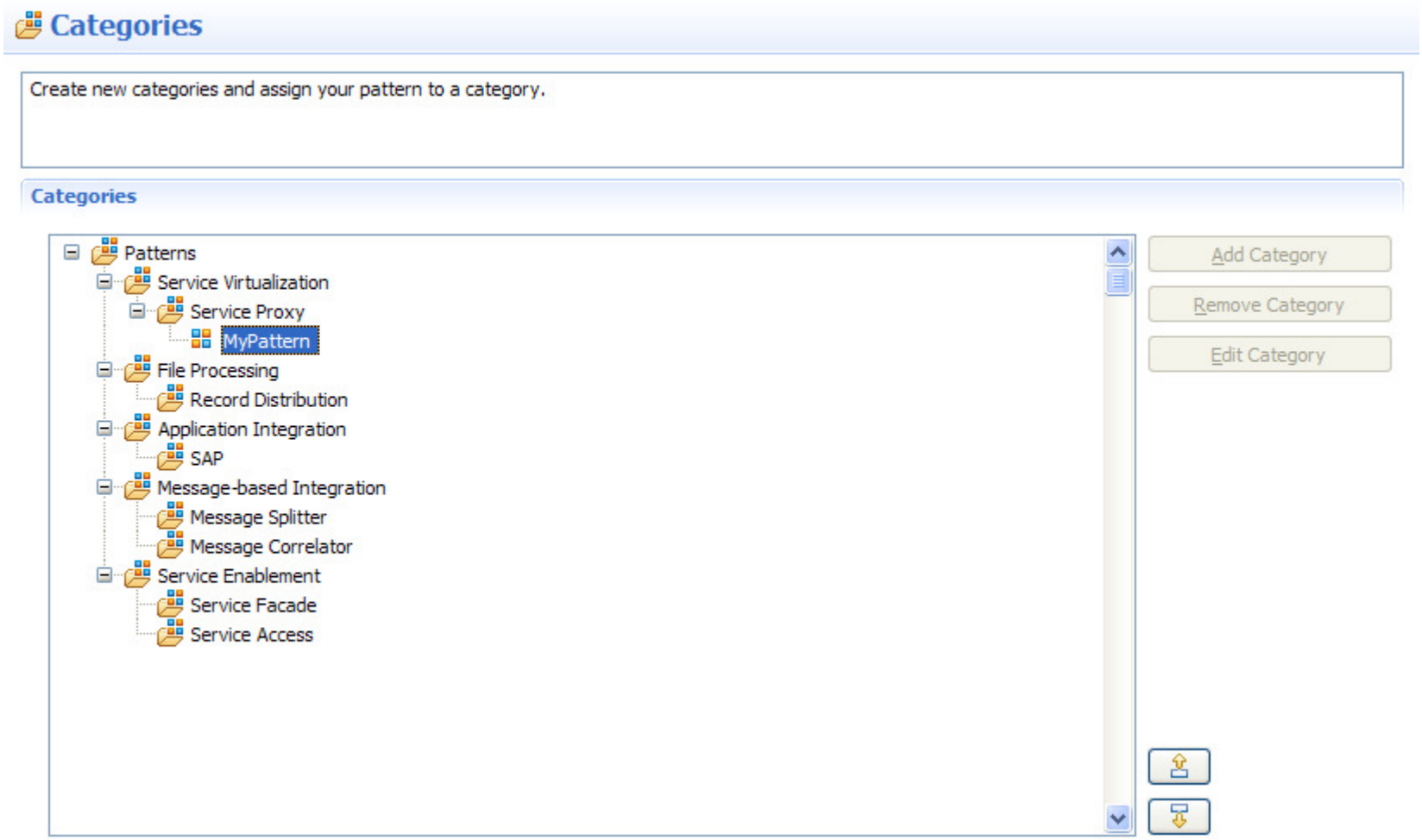
# Configure Pattern Parameters (After)



# Design Points

- Every file containing one or more target properties has a pattern parameter group created automatically
- Likewise, every target property has a pattern parameter created automatically
- The pattern is always in a valid state even with no refinement
- Pattern parameter groups can be added, deleted and edited as required
  - The default name for a pattern parameter group is based on the file name
  - Only empty pattern parameter groups can be deleted
- Pattern parameters can be added, deleted and edited as required
  - The default name for a pattern parameter is based on the target property name
  - Only empty pattern parameters can be removed – target properties cannot float free, they must always be mapped to a pattern parameter
- The 1:1 mapping between pattern parameters can be changed by shuffling the target properties around (drag and drop)

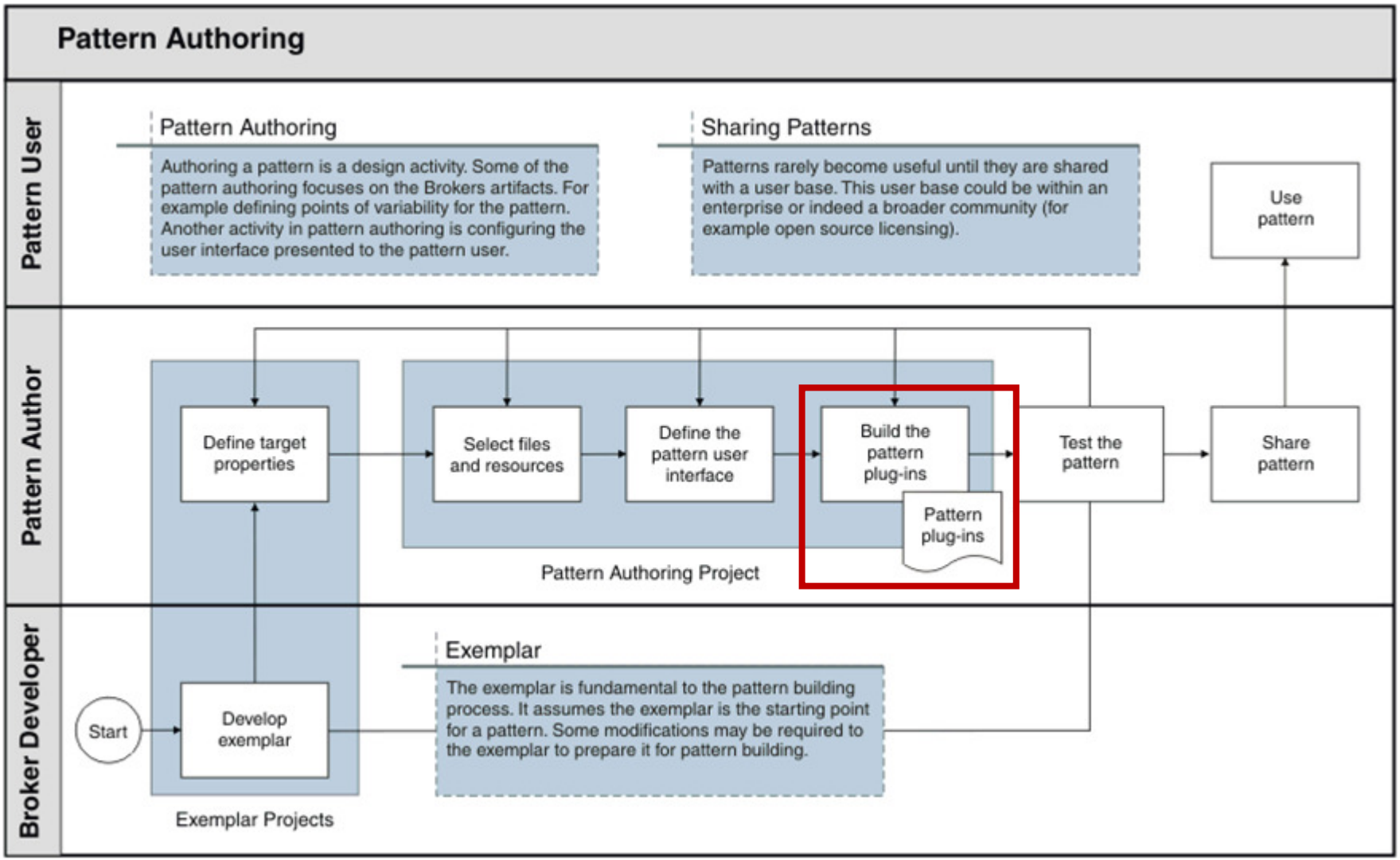
# Add Pattern Categories



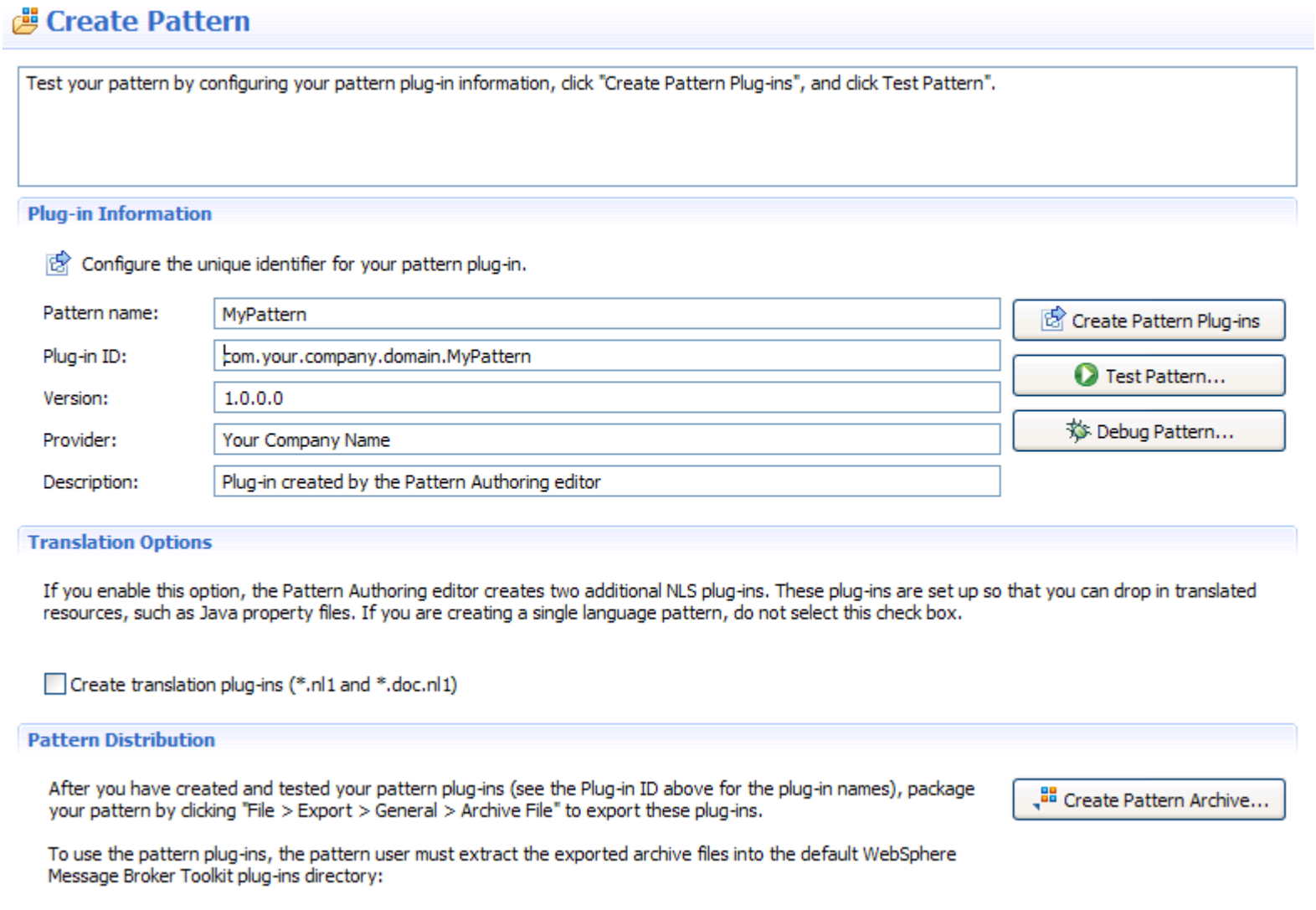
# Design Points

- Category specification is typically one HTML file
- Each category gets a directory to store its specification
- Pattern Authoring editor creates a skeleton file in the directory
- Pattern author can use any HTML editor they choose
- Specification is packaged into the pattern plug-ins when they are created
- By convention the category specification is called `overview.htm`
- All files and sub-directories are packaged as well
  - For example, to brand the category with images and stylesheets (CSS)
- There are no restrictions on where new categories are added
  - The pattern can be added to any category either existing or new

# Build The Pattern Plug-ins



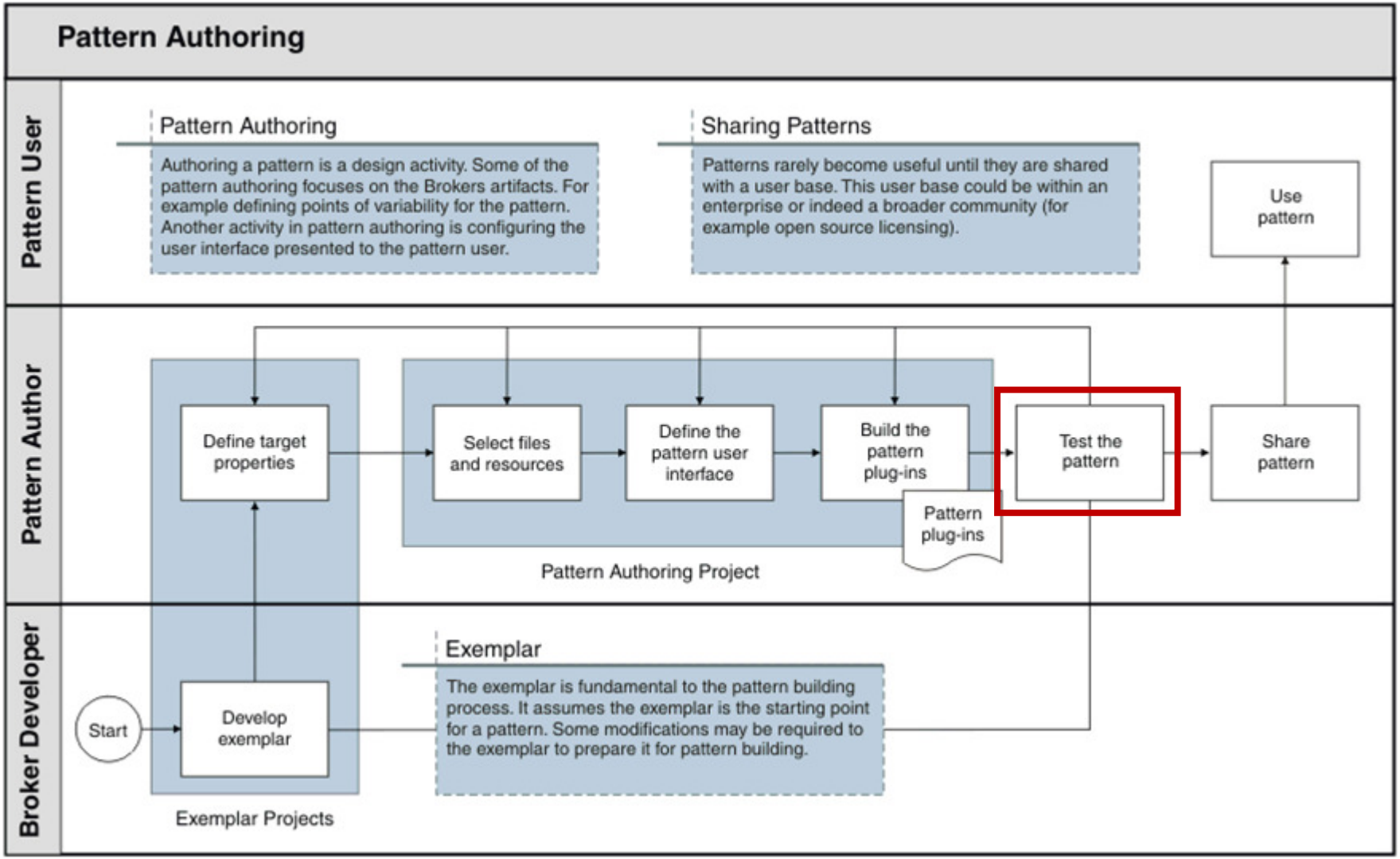
# Build The Pattern Plug-ins



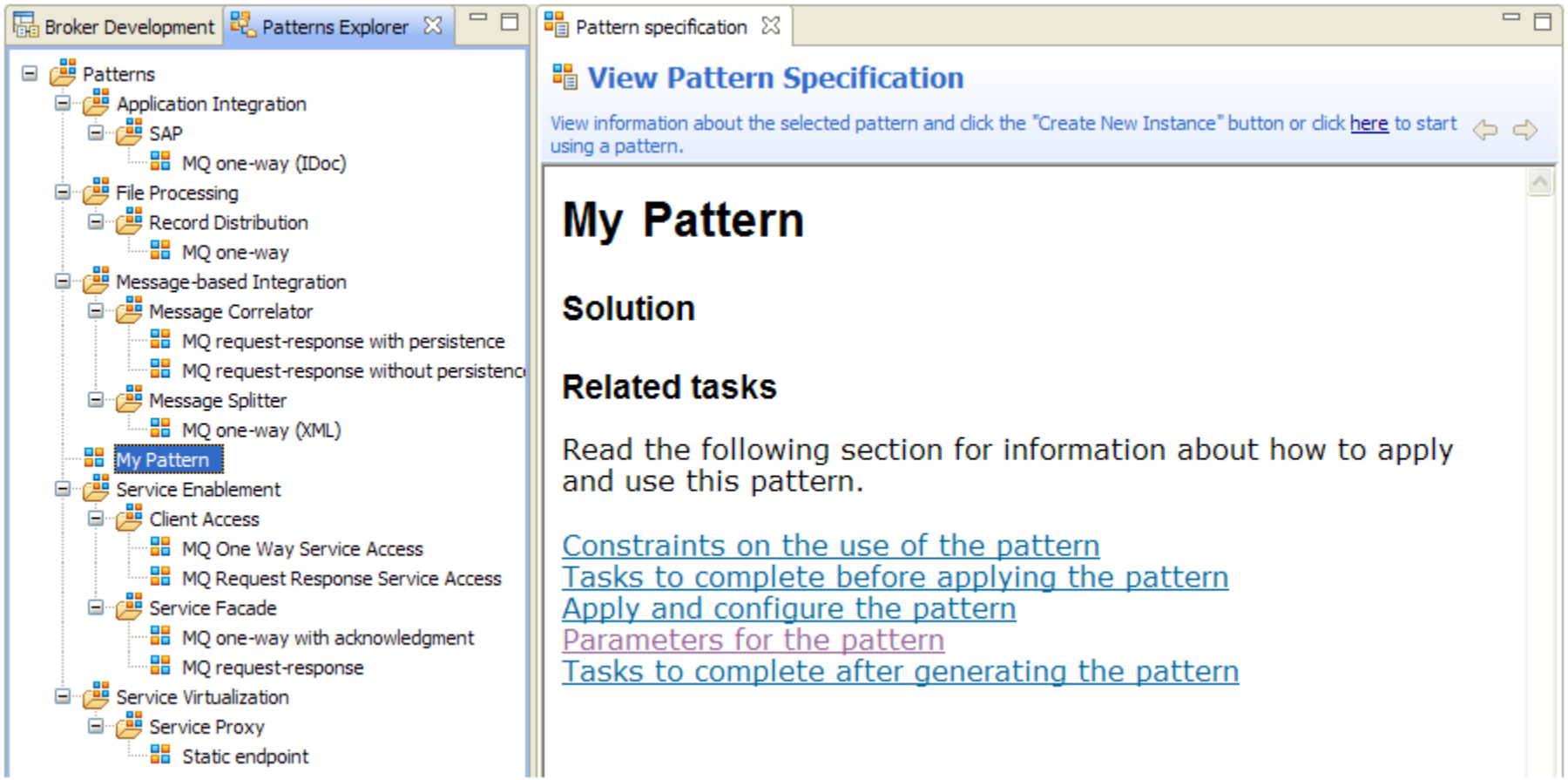
# Design Points

- The Pattern Authoring Editor creates Eclipse plug-ins
  - Eclipse provides excellent support for plug-ins and features for distribution
- Some plug-in information is required such as provider and version
- It is not recommended to edit the generated plug-ins
  - One exception is to add translated property files into the NLS plug-ins

# Test The Pattern

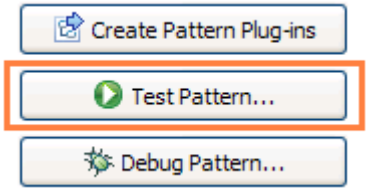


# Test The Pattern



# Design Points

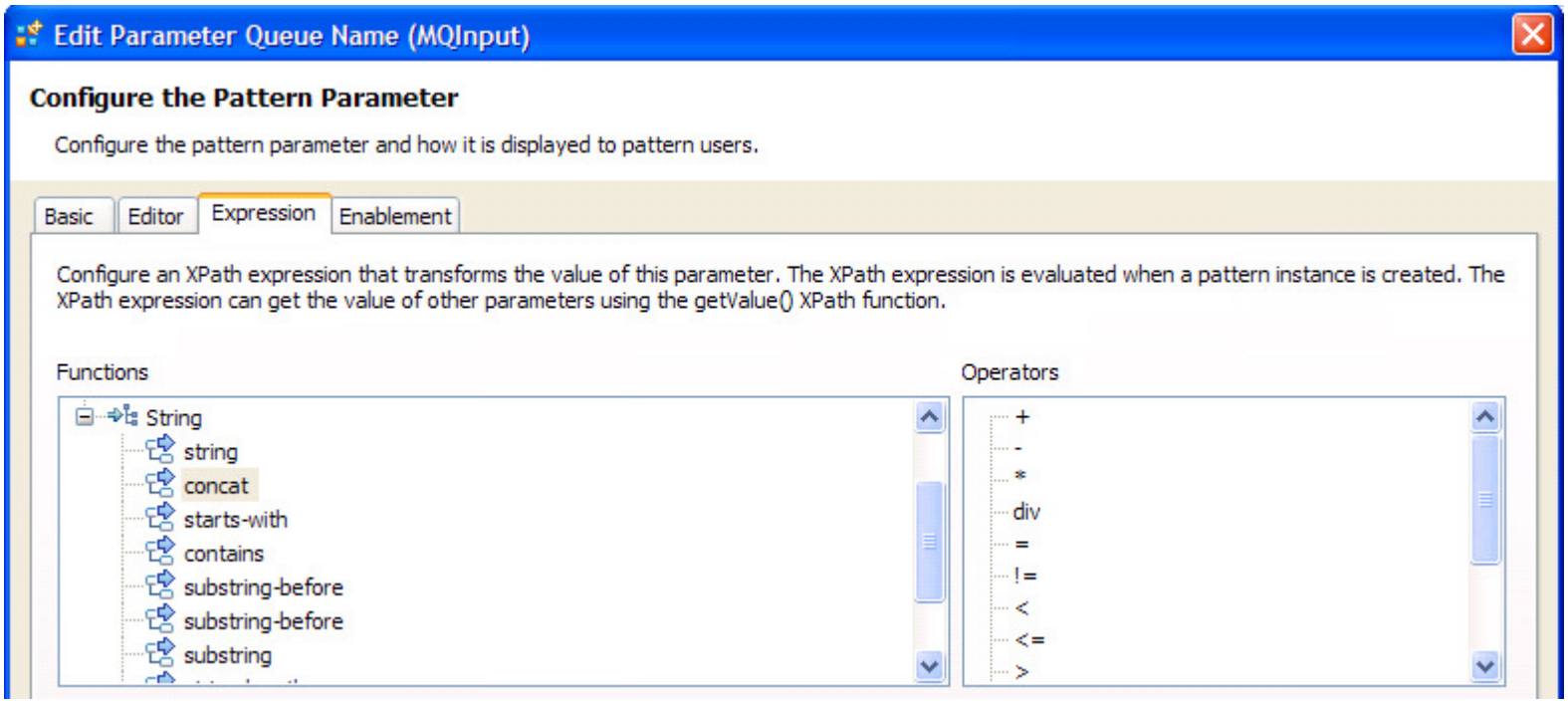
- Testing a pattern requires a re-launch of the Toolkit
  - Run configurations are an often misunderstood area of Eclipse development so the Pattern Authoring Editor provides a simple way to re-launch the toolkit:




- The pattern appears in the Pattern Explorer and is ready to test
  - The Pattern Authoring Editor will allow new categories to be created
  - A skeleton pattern specification is created which can also be changed
- This step completes the application development cycle for a pattern
  - In practice a pattern author would loop around these steps many times!

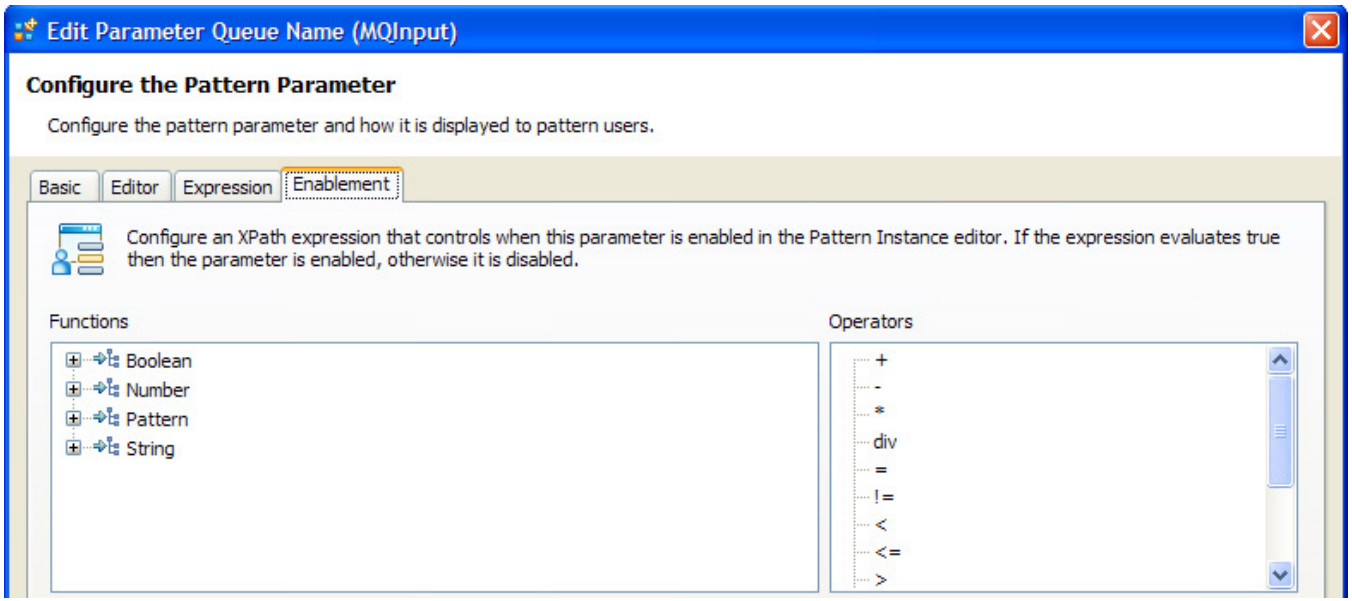
# Transformation

- The Pattern Authoring editor uses XPath as its expression language
  - XPath is a general purpose expression language!
- XPath expressions can transform pattern parameter values:
  - Pattern authors can configure an XPath expression for a pattern parameter
  - Expressions are evaluated when the pattern instance is generated



# Enablement

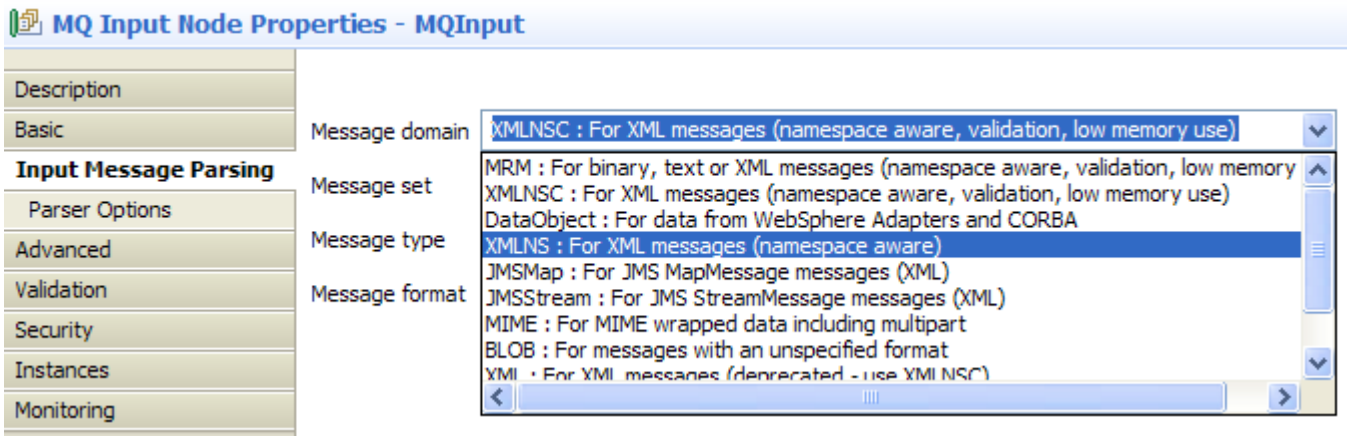
- Enablement uses an XPath expression to control when a pattern parameter is enabled in the Pattern Instance editor




- The enablement expression is evaluated every time a pattern parameter referenced in the XPath expression changes value
- If the result of the evaluation is `true` then the editor is enabled

# Enumerated Types

- Target properties are strongly typed
  - String, integer, boolean and enumerations and the most common types
  - An enumeration is a list of permissible values for a given property:




- The Pattern Authoring editor has full support for enumerations
  - An enumerated type is automatically created when a target property is added
  - The enumerated type includes the display names and property values
  - The list of values presented to the pattern user can be reduced if required

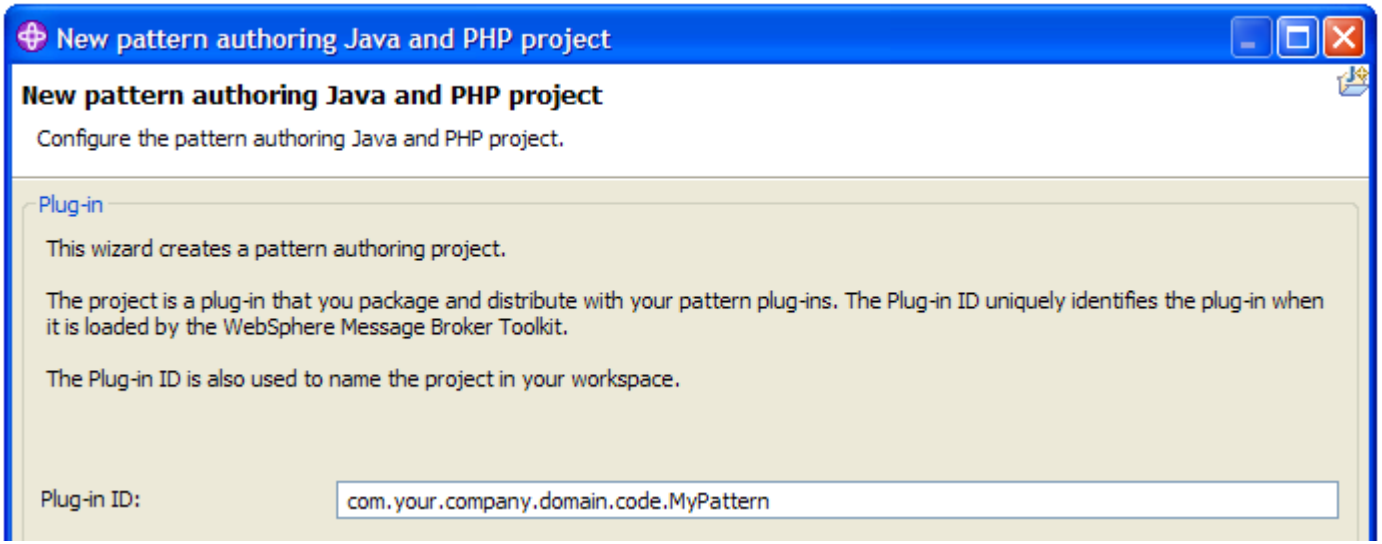

# Pattern Refinement

# Overview

- Pattern authoring in Message Broker FP1 supports property changes (node, UDP and promoted node properties)
- Property variability is the most common type of variability that a pattern might need to express - there are many others:
  - Generate application text files such as ESQL scripts
  - Make structural changes to Message Flows
  - Create administration files such as MQSC scripts
- It is impossible to try and predict all the possible extensions that a pattern author might wish to implement
- In Message Broker FP2 we provide two ways to extend pattern authoring
  - Java code that is invoked when pattern instances are generated
  - PHP templates that generate text files in pattern instance projects

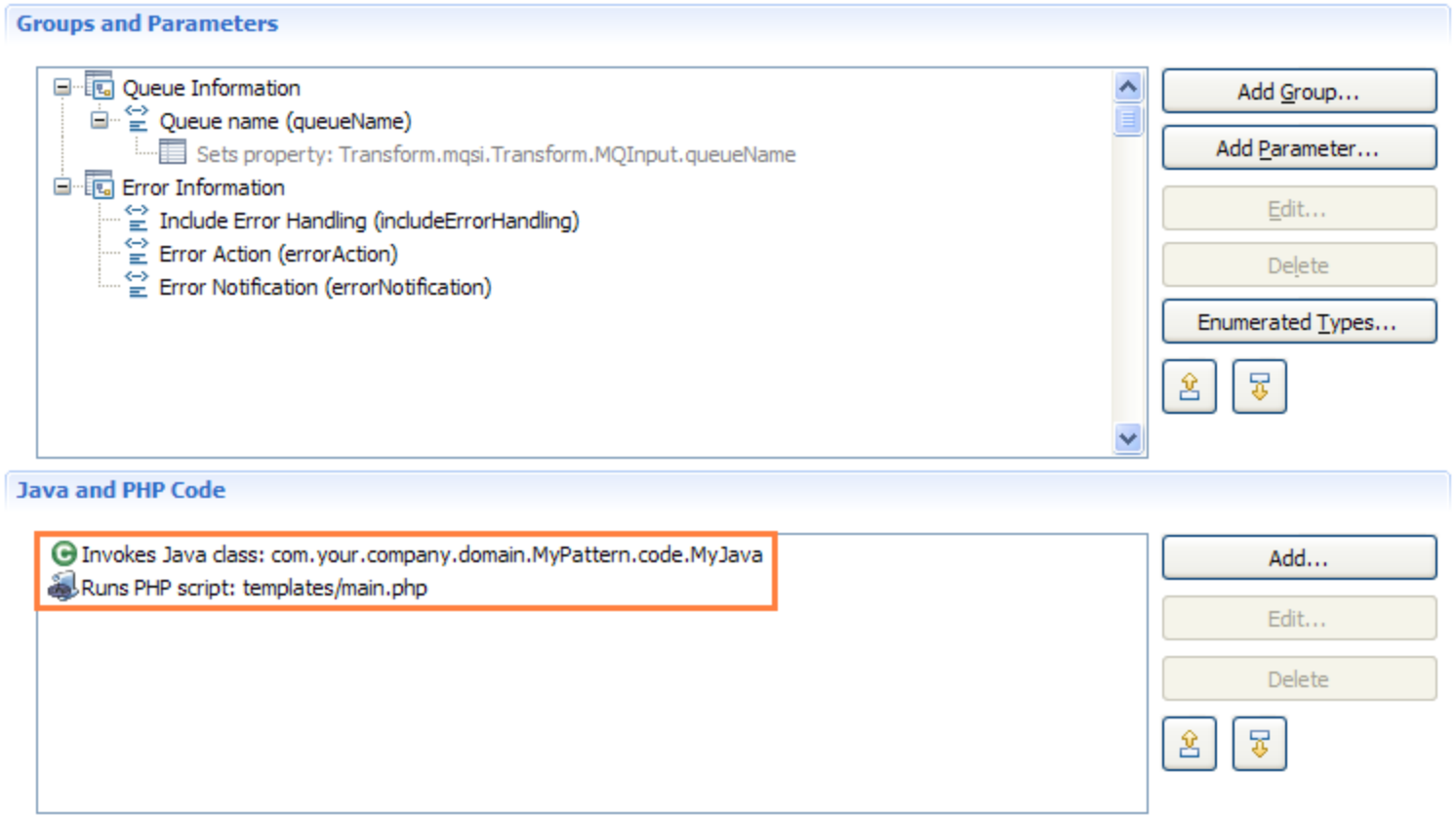
# Where Does Your PHP and Java Code Go?

- Pattern refinements are packaged in one or more *separate* plug-ins
  - Packaged and distributed with the generated pattern plug-ins
  - Clean separation between code written by the pattern author and the plug-ins generated by the Pattern Authoring editor
  - Makes it very easy to re-use Java and PHP between patterns
  - Straightforward to version, patch and upgrade the plug-ins
  - All the plug-ins can still be packaged in a single ZIP file for distribution
- New pattern authoring wizard make it a breeze to create plug-ins!



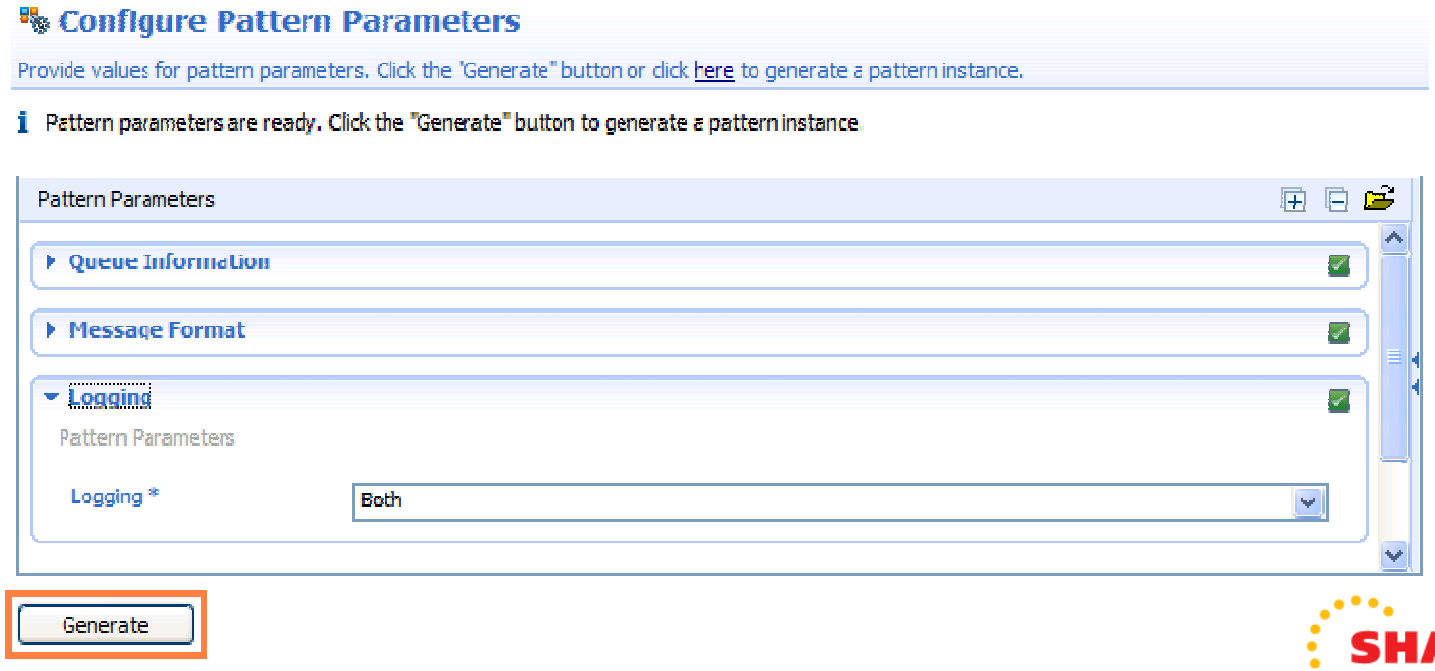
# Java and PHP code

- Easy to attach Java and PHP code to a pattern – the code is invoked in top down order as shown in the Pattern Authoring editor



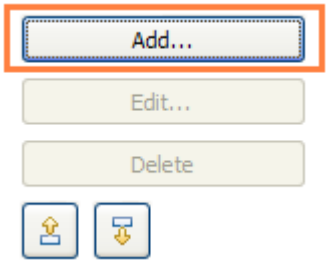
# Pattern Instance Generation

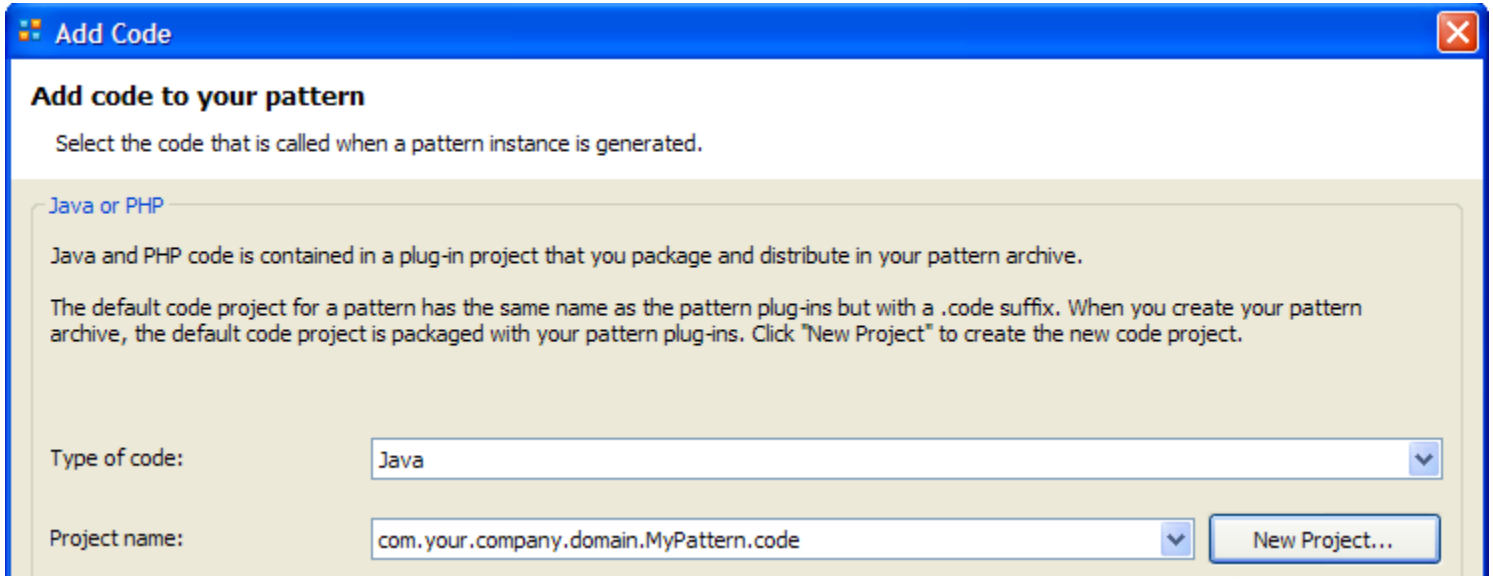
- The following sequence of actions generates pattern instance projects:
  1. The pattern authoring runtime creates the pattern instance projects
  2. All non message flow files are copied into the pattern instance projects
  3. The message flows are loaded into memory from the pattern plug-ins
  4. All target properties are configured in the message flows
  5. PHP and Java targets are invoked in top-to-bottom order
  6. The message flows are saved into the pattern instance projects



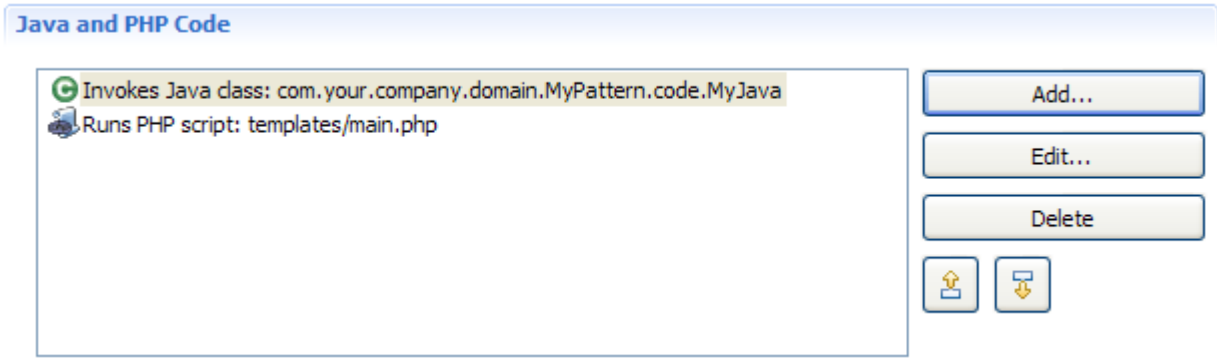
# Java Code

- Java classes implement `GeneratePatternInstanceTransform`
- The pattern authoring runtime invokes on`GeneratePatternInstance`
- The method is passed a reference to a `PatternInstanceManager`
  - Provides access to pattern parameters, pattern instance name and workspace location
  - Most important of all, the Java class can manipulate message flows!



# Java Example




```
package com.your.company.domain.MyPattern.code;

import com.ibm.broker.config.appdev.patterns.GeneratePatternInstanceTransform;
import com.ibm.broker.config.appdev.patterns.PatternInstanceManager;

public class MyJava implements GeneratePatternInstanceTransform {

    @Override
    public void onGeneratePatternInstance(PatternInstanceManager patternInstanceManager) {

        // The location for the generated projects
        String location = patternInstanceManager.getWorkspaceLocation();

        // The pattern instance name for this generation
        String patternInstanceName = patternInstanceManager.getPatternInstanceName();

    }
}
```

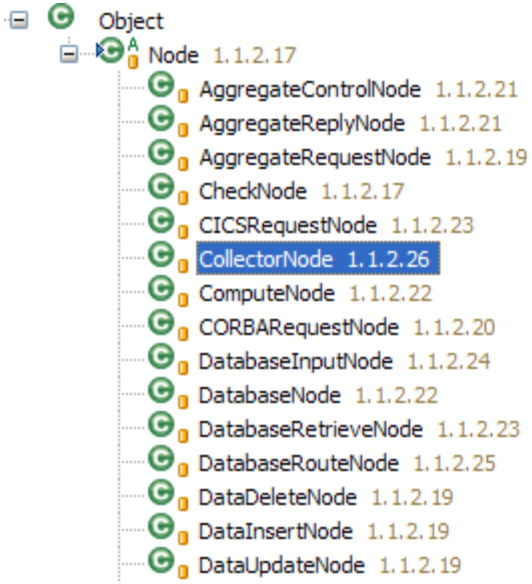
# Manipulating Message Flows

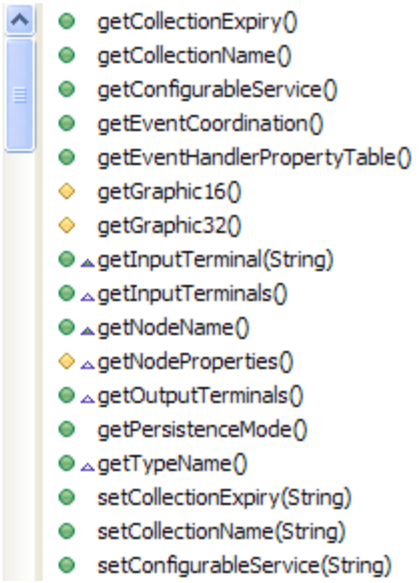
- Retrieve a message flow using the `patternInstanceManager`
- Pass the (exemplar) project name and the relative path to the file
- The message flows are loaded into memory from the pattern plug-ins
  - The flows are automatically saved into the pattern instance projects at the end
- Your changes act on the message flow immediately

```
public class MyJava implements GeneratePatternInstanceTransform {

    @Override
    public void onGeneratePatternInstance(PatternInstanceManager patternInstanceManager) {
        MessageFlow transformFlow = patternInstanceManager.getMessageFlow("Transform", "mqsi/Transform.msgflow");
    }
}
```

# Message Flow API

- The `MessageFlow` API contains all the nodes, connections and artefacts present in the message flow

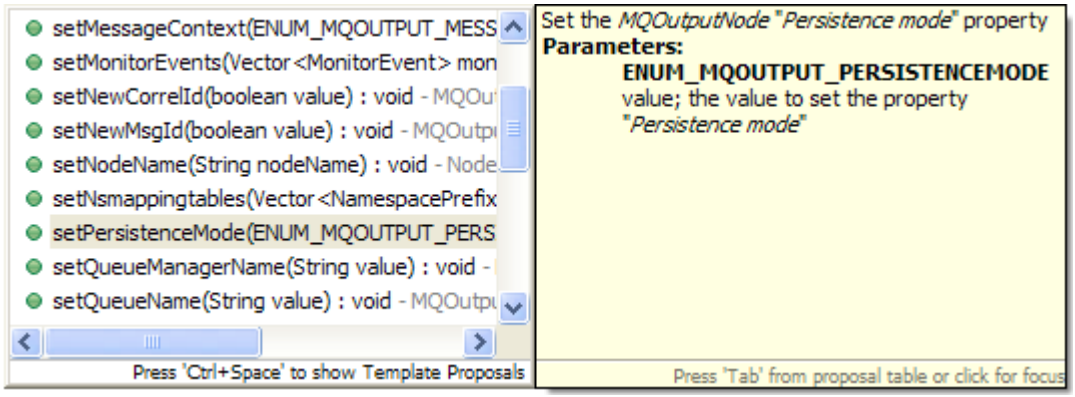



- The Message Flow API includes properties, nodes and connections
  - This is not an exhaustive list!
- User Defined Nodes are represented as `GenericNode`

# Message Flow API

- Helper methods throughout the Message Flow API
  - For example, finding a node by its label using `getNodeByName`
- Node properties are strongly typed wherever possible
- Enumerations are provided for properties
  - Implemented as type-safe classes
- JavaDoc provided for properties (easily viewed in the Java editor)



# Creating Connections Between Nodes

- Nodes can be connected using terminals (in the `Node` class)

```
                    OutputTerminal outputTerminal, InputTerminal inputTerminal
transformFlow.connect(outputTerminal, inputTerminal);
```

**Connection com.ibm.broker.config.appdev.MessageFlow.connect(OutputTerminal outputTerminal, InputTerminal inputTerminal)**

Connects the output terminal of one node into the input terminal of another node, and stores the association inside the message flow. If the affected nodes are not already part of the message flow, they will be automatically added to the message flow as part of this call. If a connection already exists between the same terminal instances, this method does not modify the message flow and the existing Connection is returned.

**Parameters:**

**outputTerminal** Output terminal of the source node

**inputTerminal** Input terminal of the target node

**Returns:**

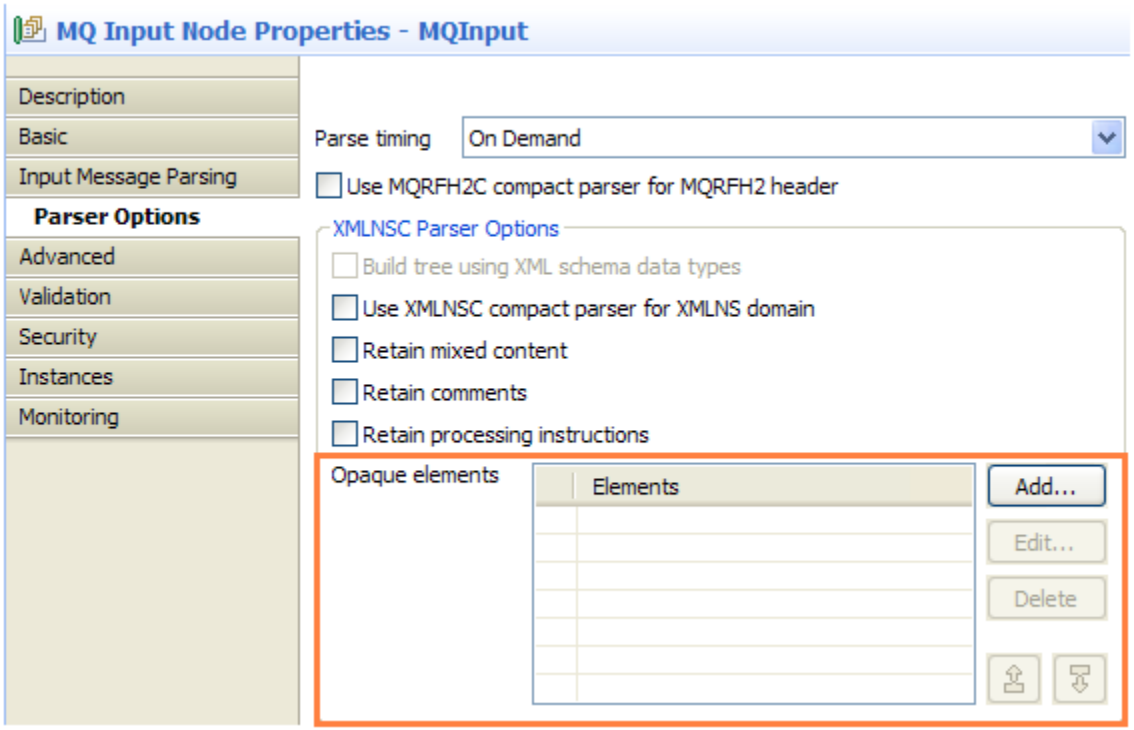
Connection The new Connection that was formed, or the existing Connection if one already existed between the supplied terminals.

- Dynamic terminals and `GenericNode` terminals also supported

```
@Override
public void onGeneratePatternInstance(PatternInstanceManager patternInstanceManager) {
    MessageFlow transformFlow = patternInstanceManager.getMessageFlow("Transform", "mqsi/Transform.msgflow");
    MQOutputNode outputNode = (MQOutputNode) transformFlow.getNodeByName("MQOutput");
    FileInputNode fileNode = (FileInputNode) transformFlow.getNodeByName("Read File");
    transformFlow.connect(fileNode.OUTPUT_TERMINAL_OUT, outputNode.INPUT_TERMINAL_IN);
}
```

# Complex Properties

- Complex properties are manipulated using objects (tables and rows)
- Property values on rows are strongly typed
- Rows can be added and removed from tables



# Complex Properties Example

```
package com.your.company.domain.MyPattern.code;

import com.ibm.broker.config.appdev.MessageFlow;
import com.ibm.broker.config.appdev.nodes.MQInputNode;
import com.ibm.broker.config.appdev.nodes.MQInputNode.OpaqueElementsRow;
import com.ibm.broker.config.appdev.nodes.MQInputNode.ParserXmlnscOpaqueElementsTable;
import com.ibm.broker.config.appdev.patterns.GeneratePatternInstanceTransform;
import com.ibm.broker.config.appdev.patterns.PatternInstanceManager;

public class MyJava implements GeneratePatternInstanceTransform {

    @Override
    public void onGeneratePatternInstance(PatternInstanceManager patternInstanceManager) {
        MessageFlow transformFlow = patternInstanceManager.getMessageFlow("Transform", "mqsi/Transform.msgflow");

        MQInputNode inputNode = (MQInputNode) transformFlow.getNodeByName("MQInput");
        ParserXmlnscOpaqueElementsTable table = inputNode.getParserXmlnscOpaqueElementsTable();

        OpaqueElementsRow row = table.createRow();
        row.setElements("/Invoices/Billing");

        table.addRow(row);
    }
}
```

**ParserXmlnscOpaqueElementsTable**
**com.ibm.broker.config.appdev.nodes.FileInputNode.getParserXmlnscOpaqueElementsTable()**

Retrieve the Opaque elements table for the node *FileInputNode*

***Returns:***
*ParserXmlnscOpaqueElementsTable instance which contains OpaqueElementsRow rows*

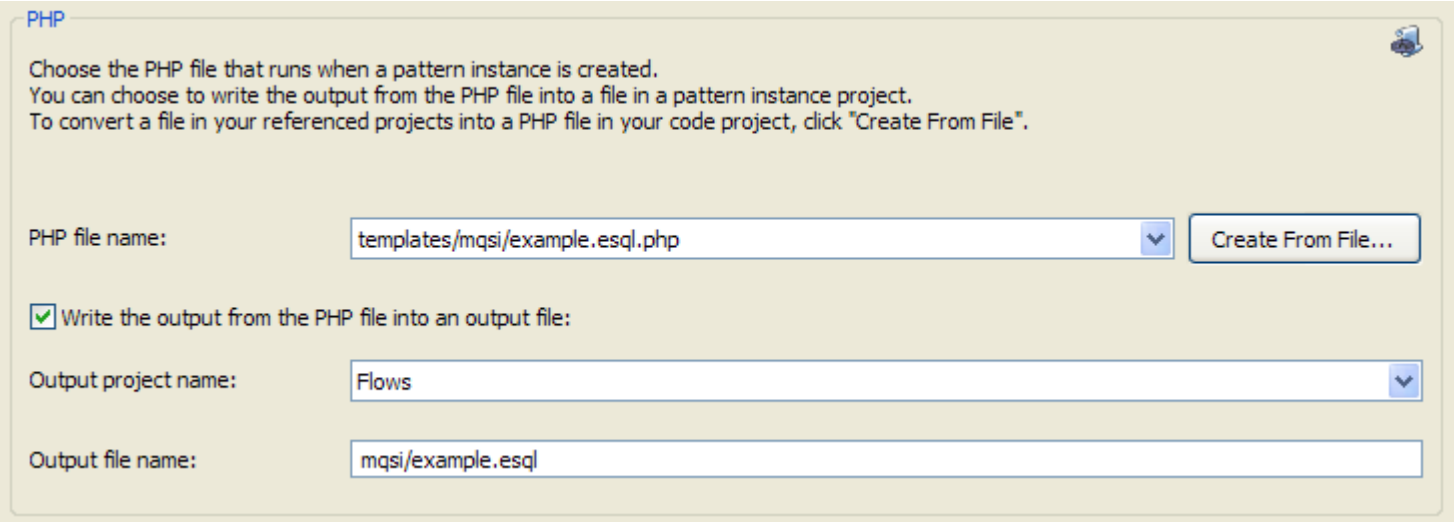
# PHP


- PHP is a general purpose dynamic scripting language
- Easy to use with a gentle learning curve
- Efficient syntax and library have evolved in open source
  - Community driven to get more done in less time
  - Impressive results with little code
  - Extensive library support
  - Language well suited to rapid prototyping


- More than 3 million developers worldwide
- 4th most populate language after Java, C and Visual Basic
- Message Broker PHP Compute Node since v6.1.0.4
  - Java implementation, fully compliant with PHP version 5.2

# PHP Templates

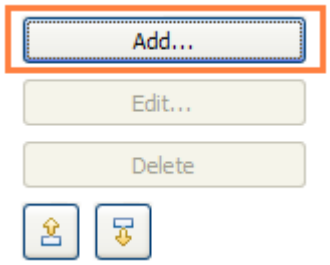
- Output from PHP template is written straight to a pattern instance file

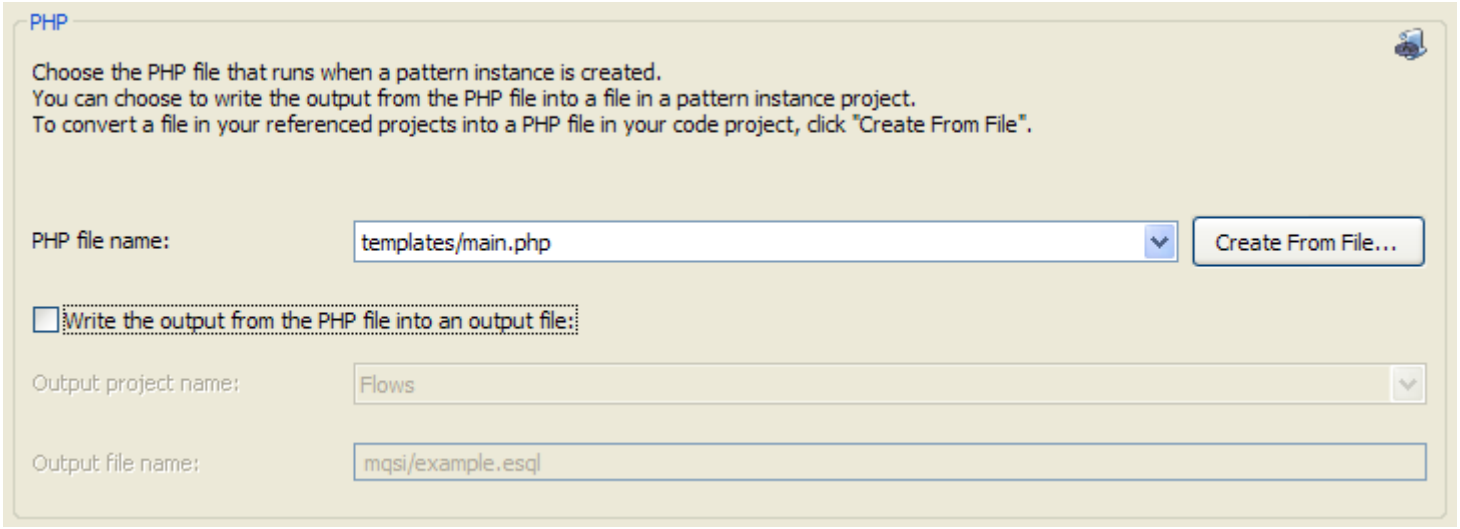



```
DEFINE QLOCAL(LOCAL) DEFPSIST(YES) DESCR('<?php echo $_MB['PP']['queueName']; ?>')
```

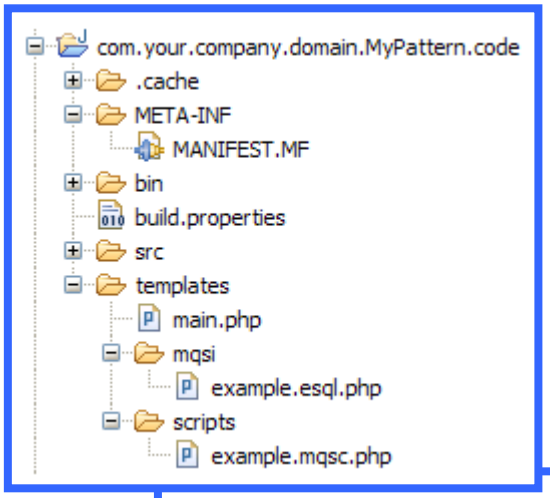
# PHP Scripts

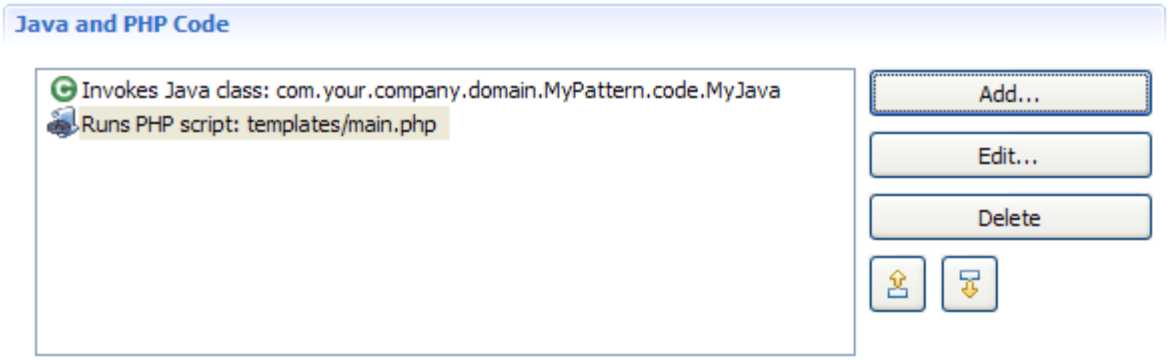
- The PHP script invokes other PHP templates to output files
  - Script acts as a controller and runs other templates as required
- Typically used when custom logic is required with pattern parameters
- Output from the controller script is sent to the Console view



# PHP Example (`main.php`)




```php
<?php

    phpinfo();

    $today = getdate();
    print_r($today);

    var_dump($_MB);

    if ($_MB['PP']['includeErrorHandling'] == 'true') {
        mb_pattern_run_template("Transform", "mqsi/example.esql.php", "mqsi/example.esql");
    }
?>
```

# PHP Example (example.esql.php)

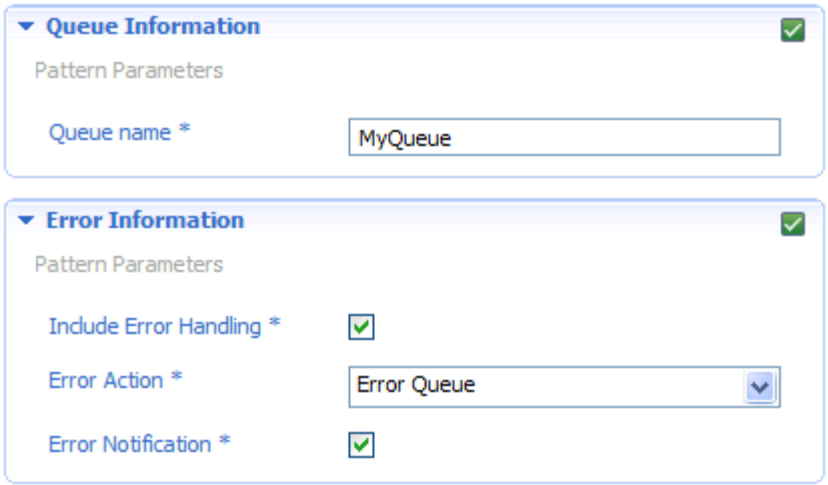
```
BROKER SCHEMA mqsi
<?php
    if ($_MB['PP']['errorAction'] == 'errorQueue') {
        echo "DECLARE ErrorAction EXTERNAL CHARACTER '".$_MB['PP']['errorAction']."';";

echo <<<ESQL

CREATE FILTER MODULE CheckErrorAction
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        IF ErrorAction = 'errorQueue' THEN
            RETURN TRUE;
        ELSE
            RETURN FALSE;
        END IF;
    END;
END MODULE;

ESQL;
    }
?>
```

This uses the PHP `echo` function to write the text into the output file - the embedded PHP code is accessing pattern parameters through the `_MB` super global

This section of ESQL script is conditionally output depending on the value of the `errorAction` pattern parameter - the surrounding PHP uses a HEREDOC to output the code as a literal string

# Pattern Authoring PHP Super Globals

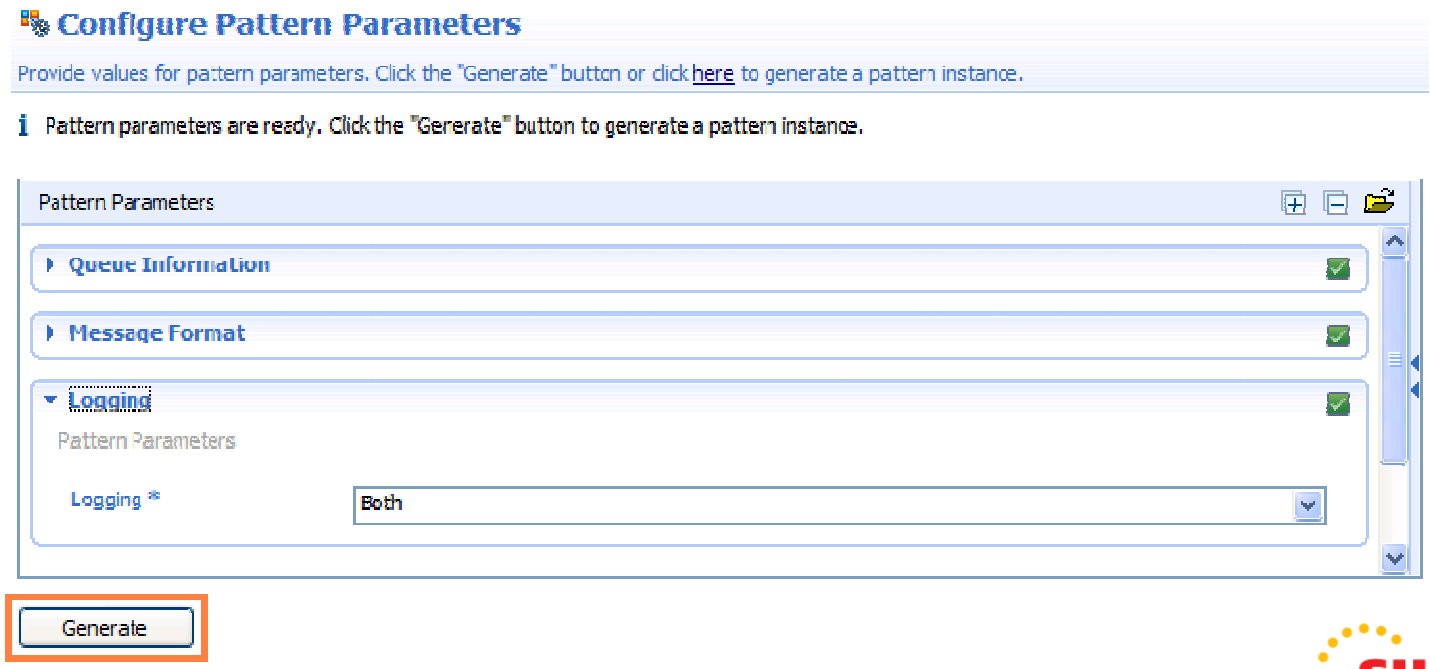
- PHP provides a large number of predefined variables to any script which it runs – Message Broker has an additional super global called `_MB`
- `_MB` in pattern authoring is populated with the pattern parameters, the pattern instance name and the `PatternInstanceManager`:




```
array(4) {
  ["PATTERN_INSTANCE_MANAGER"]=>
  object(Java)#1 (0) {
  }
  ["PP"]=>
  array(4) {
    ["queueName"]=>
    string(3) "MyQueue"
    ["includeErrorHandling"]=>
    string(4) "true"
    ["errorAction"]=>
    string(10) "errorQueue"
    ["errorNotification"]=>
    string(4) "true"
  }
  ["PATTERN_INSTANCE_NAME"]=>
  string(3) "foo"
  ["WORKSPACE_ROOT"]=>
  string(28) "D:/WMB/Workspace/TestPattern"
}
```

# Editing PHP On-The-Fly

- You do not need to re-launch a workbench to test your PHP changes
- Simply save the file in the main workbench and click **Generate** again
- This edit and test efficiency is a major productivity boost
- If you change anything else in the pattern then you need to re-launch



# PHP and Java Integration

- It is straightforward to invoke Java code from your PHP scripts:

```php
<?php
    if ($_MB['PP']['logging'] == 'both') {
        $pim = $_MB["PATTERN_INSTANCE_MANAGER"];
        $class = $pim->getPluginClass("com.your.company.domain.MyPattern.code",
            "com.your.company.domain.MyPattern.code.MyJava");
        $class->doSomeHelpfulstuff("Hello Everyone!");
    }
?>
```

```java
public class MyJava implements GeneratePatternInstanceTransform {

    public static void doSomeHelpfulStuff(String message) {
        System.out.println("Doing some helpful stuff for the pattern [" + message + "]");
    }
```

- The Message Broker PHP runtime has a feature rich Java bridge
  http://www.projectzero.org/sMash/1.1.x/docs/zero.devguide.doc/zero.php/ZeroAdvancedPHPJavaBridge.html
- This PHP script is manipulating the message flow using the Java API!

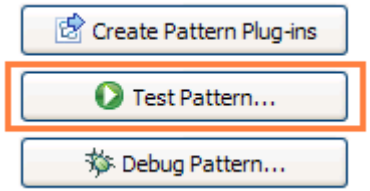
```php
<?php
    $pim = $_MB["PATTERN_INSTANCE_MANAGER"];
    $flow = $pim->getMessageFlow("Transform", "mqsi/Transform.msgflow");
    $node = $flow->getNodeByName("MQInput");
    $node->setShortDescription("Configured by PHP!");
?>
```

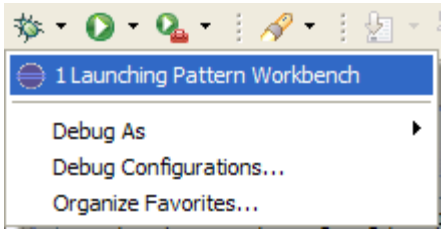
# Standard PHP Functions

- The full list of supported PHP extensions and functions are listed here:
  - http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac69026_.htm
- Extensive help on PHP is available on http://php.net
  - Individual functions can be located directly, for example: http://php.net/phpinfo
- The language reference for PHP is here:
  - http://www.php.net/manual/en/langref.php
- There are a few language differences between the Message Broker PHP and the Open Source PHP implementation – they are listed here:
  - http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp

# Debugging Your Pattern Refinements

- Java pattern refinements can be debugged in the Java debugger
- The launch configuration is automatically created the first time you test a pattern from the Pattern Authoring editor:

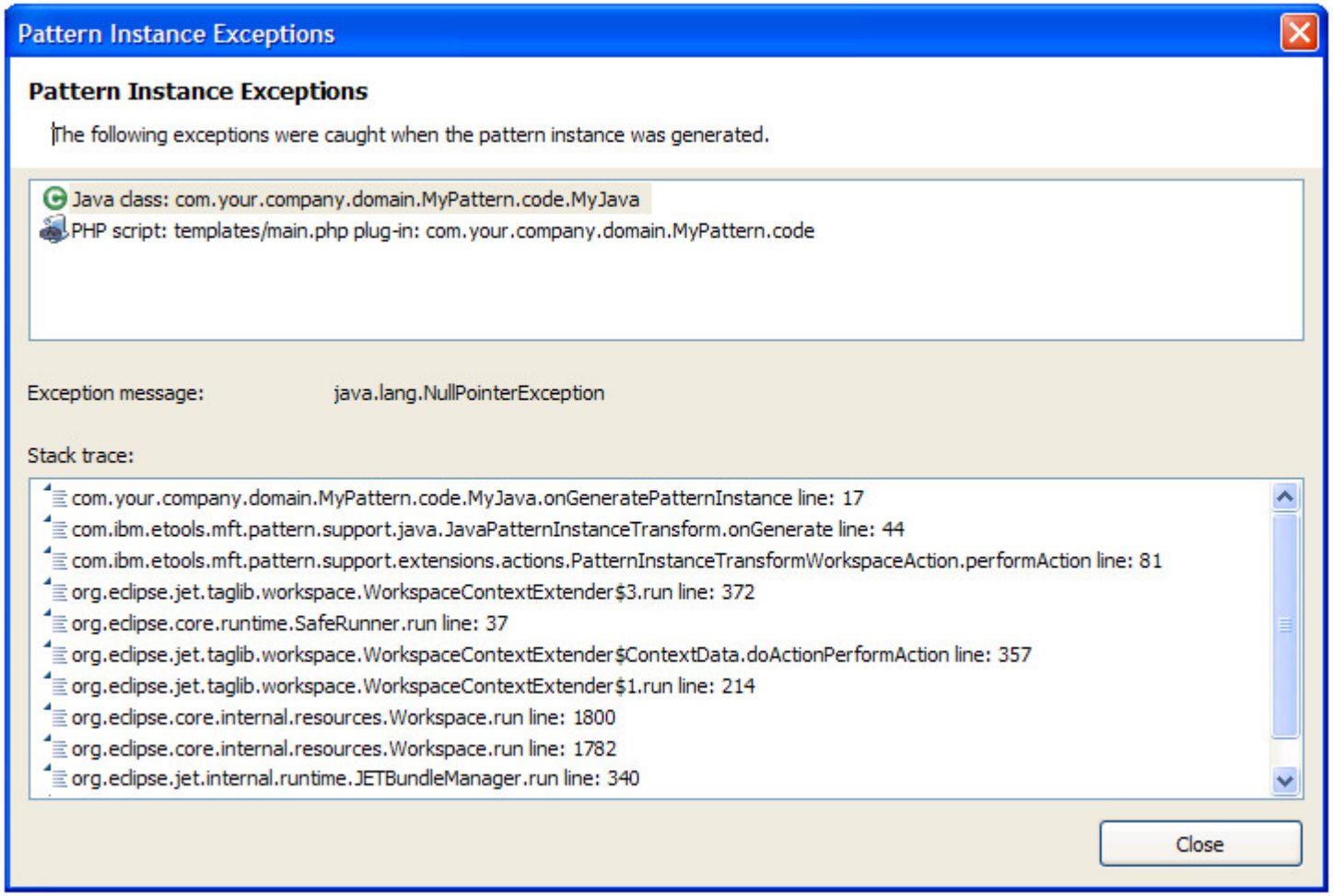



- Any console output from your PHP or Java appears in the Console view:

# Exception Handing

- Exceptions from PHP and Java code are caught and displayed



# So Which Language Should I Use?

- Because of the Java bridge in the PHP runtime, many pattern authoring tasks can be done in either language
- So here are some general recommendations – your mileage may vary!
- Use PHP for text files that need to be marked up with conditional logic
- Use Java to manipulate message flows as you get an excellent debugger and code completion in the Java editor
  - Unless you have PHP skills and prefer the productivity of scripting languages!
- If you have existing assets or libraries that you want to call, pick the language that best aligns with them

# Rolling Your Own Plug-in?

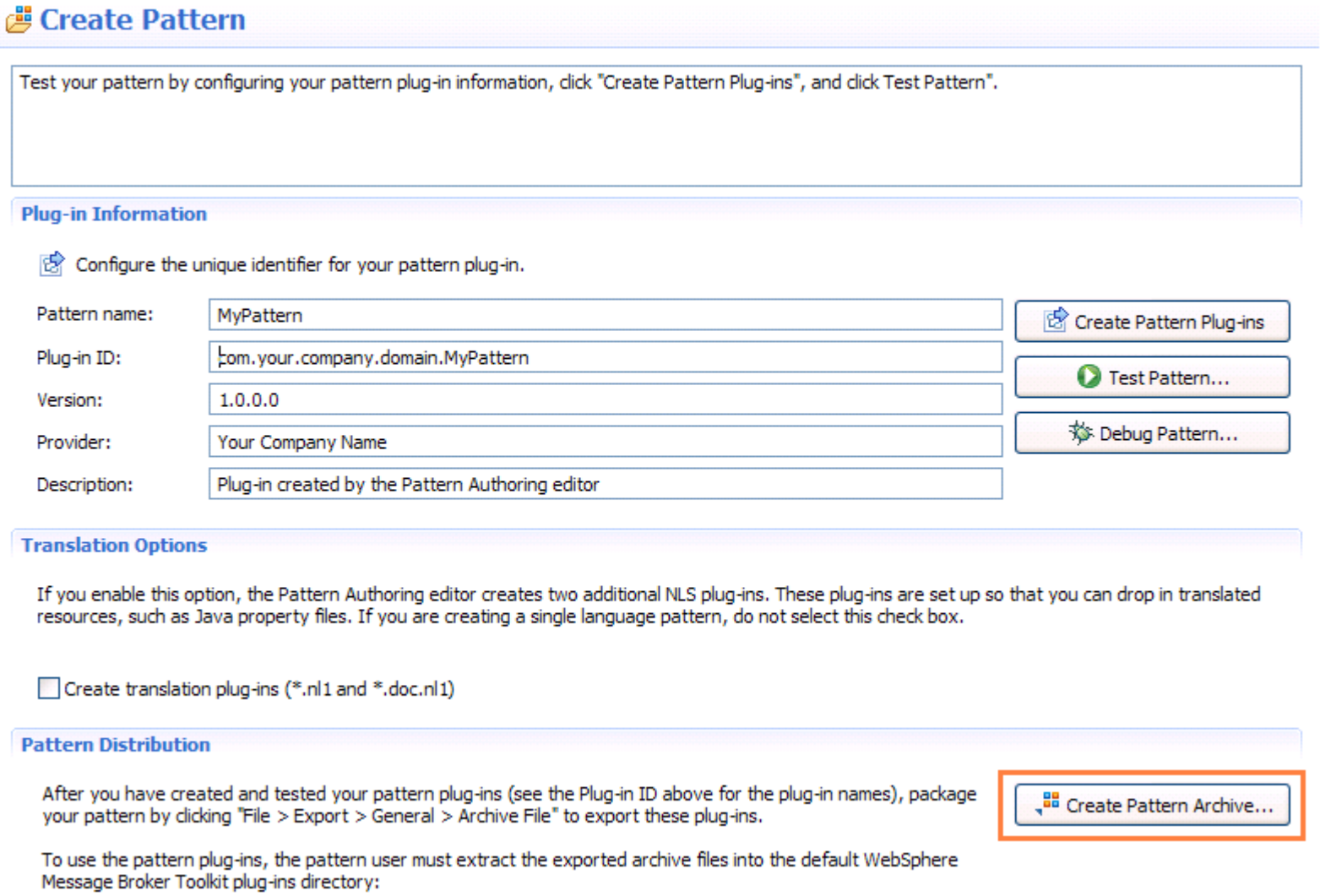
- It must be a plug-in not just a Java project!
- Your plug-in must export the Java classes that are to be invoked
- You will need a reference to `com.ibm.broker.config.appdev`
- There is a reference from the pattern plug-ins to your plug-in
  - The pattern authoring generator adds this to the plug-in manifest
- If it contains PHP templates then it must be a directory plug-in
  - PHP scripts can only be loaded direct from the file system and not from a JAR
- The plug-in must be installed before pattern instances can be created
  - So think about how you want to distribute your code!


# Pattern Communities

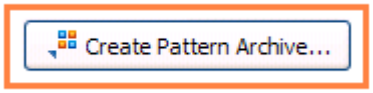
# Pattern Communities

- Pattern authoring creates patterns whose value multiplies as they are shared and used by a community of developers
- Public and private communities are equally interesting!
- So how to distribute and share patterns effectively?
- A community needs to offer more than just a repository of assets!
  - Space for content such as patterns and subflow nodes
  - Forums to host discussions and threaded conversations
  - Categorisation is important as the community grows! (taxonomies)
  - Essential administrative functions such as user management
  - News articles and broadcasts keep the site fresh
- There are literally dozens of content management systems available
- On the commercial side we have tried Lotus Connections and Rational Asset Manager
- Open source projects such as Drupal and Joomla are available

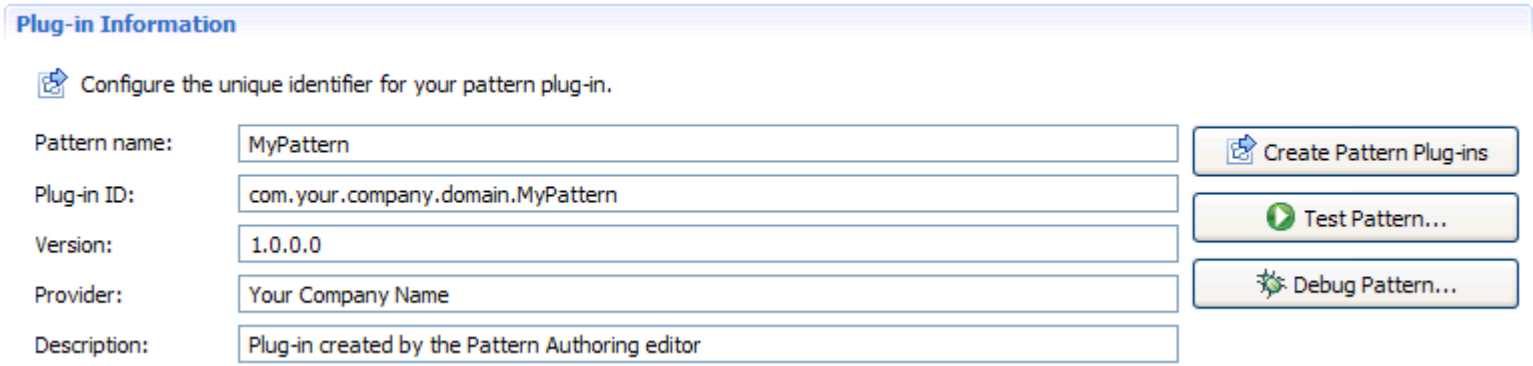
# Packaging a Pattern



# Pattern Archives




- The goal is to simplify the packaging and distribution of patterns
- Creating Eclipse update sites and features is a fairly complex task
- A pattern archive contains the pattern plug-ins and an Eclipse feature
  - The version of the feature matches the pattern version:



# Creating a Community Space

## WebSphere Message Broker

Developing patterns and subflow nodes in Message Broker

**New forum topics**

- Question about the Service Proxy pattern

more

**Active forum topics**

- Question about the Service Proxy pattern

more

**admin**

- My account
- Create content
- Administer
- Log out

Drupal

**Search**

Search this site:

Search

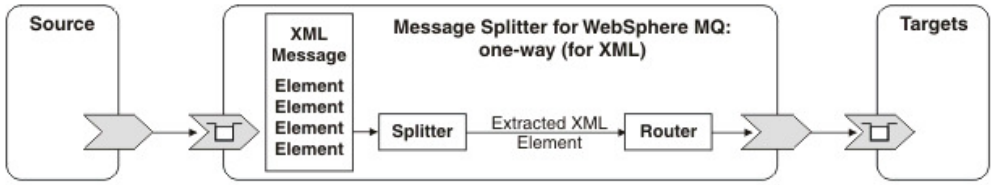
### Patterns Community

This site shows you how to build a community web site to share Message Broker patterns, subflow nodes and much more!

The web site is built using Drupal, a very popular and well established open source content management system (CMS):

- Voting is provided by **FiveStar** and **VotingAPI** modules
- Site search is configured (this is a built in module)
- An easy to use **Administration** module is available
- Pages based around an image are provided by the **Image** module
- Categorisation of content made available by the **Taxonomy** module

### Message Splitter Pattern




This is the first version of a new message splitter pattern.

Use the Message Splitter for WebSphere MQ: one-way (for XML) pattern to split a large XML message into smaller elements for processing by one or more targets, by using transactional flows and persistent WebSphere MQ messages.

**Who's online**

There are currently *1 user* and *0 guests* online.

**Online users**

- admin

**Syndicate**

**Who's new**

- admin

**Taxonomy**

- Patterns
  - Application Integration
  - Database Integration
  - Java
  - Message Based Integration
  - Message Splitter
  - PHP
  - Service Facade

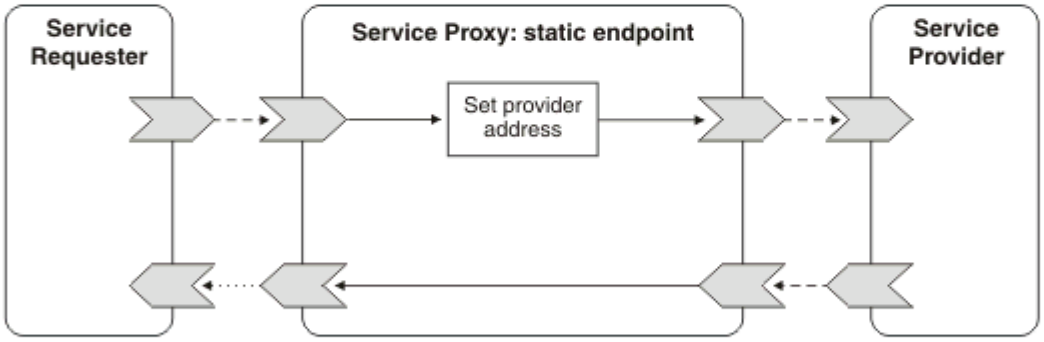
# A Shared Repository for Patterns

## Service Proxy Pattern

Use this pattern to provide decoupling between Web service requesters and Web service providers by routing through a virtual service that is bound directly to the target service provider.

Use this pattern:

1. When you do not want the client to access the service directly, either because the address of the endpoint at which the service is located might change, or because it must be hidden to control access to the service
2. To support service access over HTTP or HTTPS
3. When you want to provide logging of all requests to the service provider, or when you want to provide additional error handling capabilities without making changes to the service provider



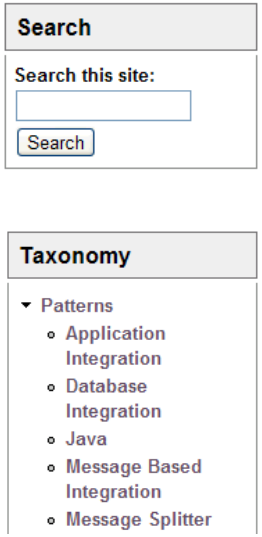

**Average:**

Your rating: 2.5 Average: 2.5 (1 vote)

| Attachment | Size |
|---|---|
| MyPattern.patternzip | 51.36 KB |

# Drupal


- The demonstration system is a standard Drupal v6 install
  - Drupal is a PHP-based open-source content management system (CMS)
  - A key strength of Drupal is its vibrant community of users and developers
  - Drupal has an extensive list of many hundreds of add-on modules
- The following contributed modules were installed:
  - Voting is provided by FiveStar and VotingAPI modules
  - Site search is configured (this is a built in module)
  - An easy to use Administration module is available
  - Pages based around an image are provided by the Image module
  - Categorisation of content made available by the Taxonomy module
- It took just over one hour to complete the system set up!



# Community Branding

**New forum topics**

Question about the Service Proxy pattern

more

**admin**

My account
Create content
Administer
Log out

**Taxonomy**

- Patterns
  - Application Integration
  - Database Integration
  - Java
  - Message Based Integration
  - Message Splitter
  - PHP
  - Service Facade
  - Service Proxy
  - User-Defined Subflow Nodes
  - Web Services
  - WebSphere MQ

## Patterns Community

Posted Sun, 09/26/2010 - 15:26 by admin

This site shows you how to build a community web site to share Message Broker patterns, subflow nodes and much more!

The web site is built using Drupal, a very popular and well established open source content management system (CMS):

- Voting is provided by FiveStar and VotingAPI modules
- Site search is configured (this is a built in module)
- An easy to use Administration module is available
- Pages based around an image are provided by the Image module
- Categorisation of content made available by the Taxonomy module

## Message Splitter Pattern

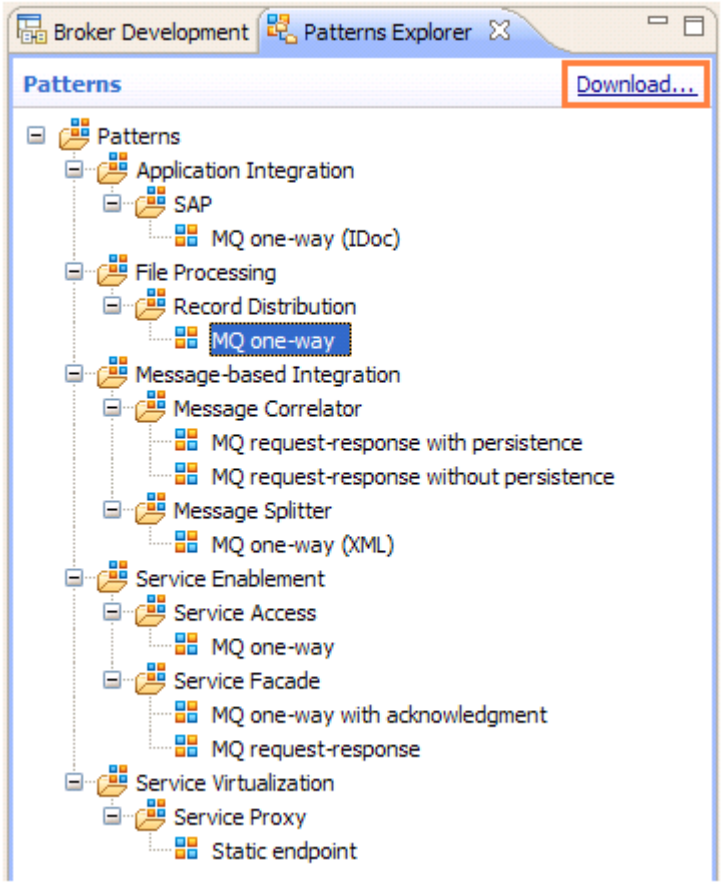
Posted Sun, 09/26/2010 - 14:24 by admin


This is the first version of a new message splitter pattern.

Use the Message Splitter for WebSphere MQ: one-way (for XML) pattern to split a large XML message into smaller elements for processing by one or more targets, by using transactional flows and persistent WebSphere MQ messages.

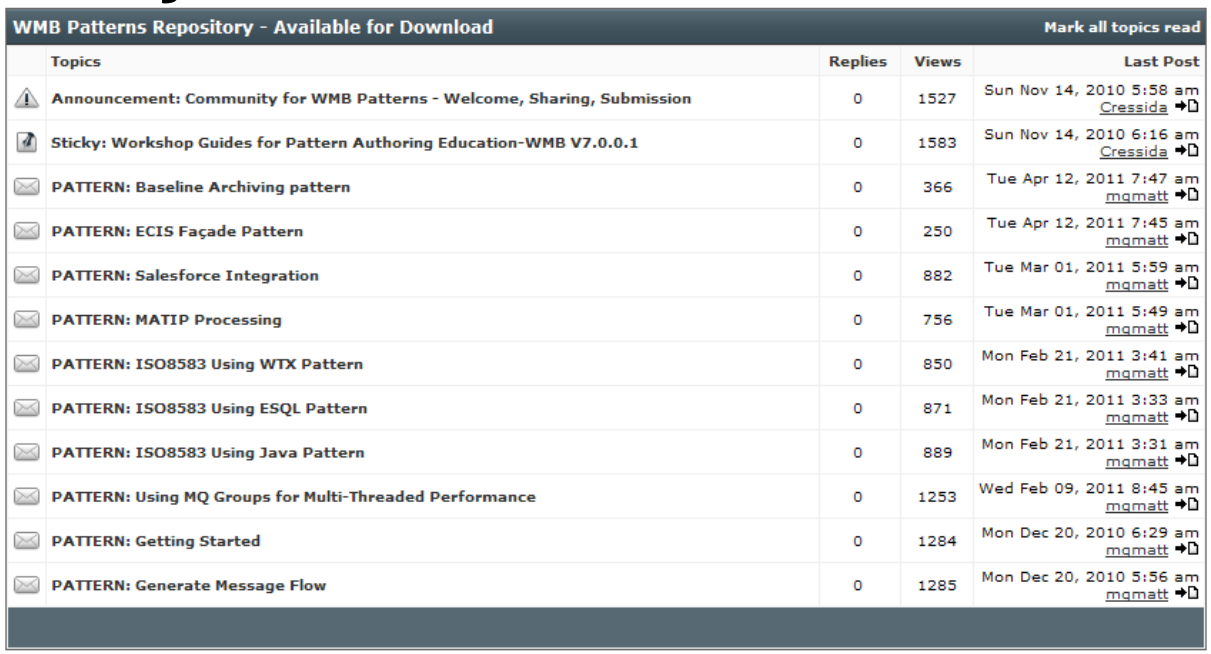
Use this pattern when you have applications that store information about a number of business transactions and transmit this information to one or more target applications in batches. It can be used to handle large messages without excessive memory use.

# Installing Patterns




- The Pattern Explorer provides a download capability for pattern archives
- Pattern archives can be installed from a local file system or from a URL
- HTTP/S is not supported – instead login, download and install from the file system
- Installation completes without a toolkit restart
- Patterns are uninstalled through the Eclipse updates feature: Help > Software Updates
- `.patternzip` files can also be double clicked which runs an installer application (the file extension is registered by the toolkit)

# Announcing mqseries.net Patterns Community



WMB Patterns Repository - Available for Download — Mark all topics read

| Topics | Replies | Views | Last Post |
|---|---|---|---|
| Announcement: Community for WMB Patterns - Welcome, Sharing, Submission | 0 | 1527 | Sun Nov 14, 2010 5:58 am Cressida |
| Sticky: Workshop Guides for Pattern Authoring Education-WMB V7.0.0.1 | 0 | 1583 | Sun Nov 14, 2010 6:16 am Cressida |
| PATTERN: Baseline Archiving pattern | 0 | 366 | Tue Apr 12, 2011 7:47 am mqmatt |
| PATTERN: ECIS Façade Pattern | 0 | 250 | Tue Apr 12, 2011 7:45 am mqmatt |
| PATTERN: Salesforce Integration | 0 | 882 | Tue Mar 01, 2011 5:59 am mqmatt |
| PATTERN: MATIP Processing | 0 | 756 | Tue Mar 01, 2011 5:49 am mqmatt |
| PATTERN: ISO8583 Using WTX Pattern | 0 | 850 | Mon Feb 21, 2011 3:41 am mqmatt |
| PATTERN: ISO8583 Using ESQL Pattern | 0 | 871 | Mon Feb 21, 2011 3:33 am mqmatt |
| PATTERN: ISO8583 Using Java Pattern | 0 | 889 | Mon Feb 21, 2011 3:31 am mqmatt |
| PATTERN: Using MQ Groups for Multi-Threaded Performance | 0 | 1253 | Wed Feb 09, 2011 8:45 am mqmatt |
| PATTERN: Getting Started | 0 | 1284 | Mon Dec 20, 2010 6:29 am mqmatt |
| PATTERN: Generate Message Flow | 0 | 1285 | Mon Dec 20, 2010 5:56 am mqmatt |

Page 1 of 1

- MQSeries.net patterns community
  - A new website featuring patterns created, rated and used by the mqseries.net community
  - A natural extension to the current help and support forums, oriented towards best practices
  - Users can download patterns, install them on their local broker, and go!
  - Double-click on an mqseries.net pattern to download into your Toolkit Patterns Explorer!

Thanks to all who have contributed to the mqseries.net community

# Questions – Please Fill Out Your Session Evaluation Form!