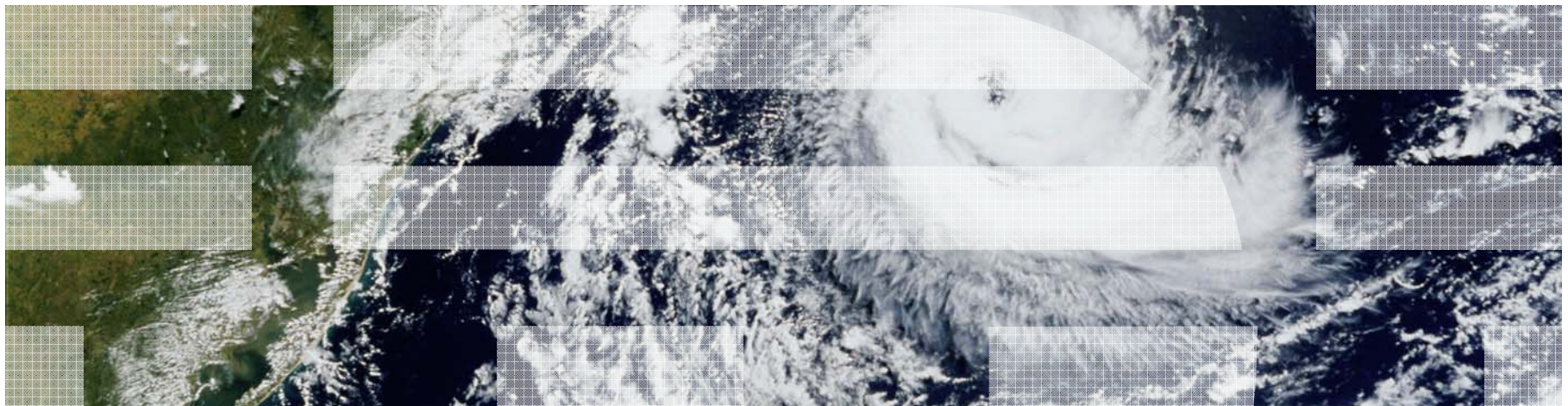


Pattern Authoring



Agenda

- Patterns refresh
- Glossary
- Pattern authoring
 - Principles
 - Workflow
 - The Exemplar
 - Variability in Patterns
 - Pattern Parameters
 - Design Walk Through
 - Customizing a pattern
- Questions

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 Broker
 - Patterns have bubbled right to the top in the navigator view!

Principles

Principles (Part I)

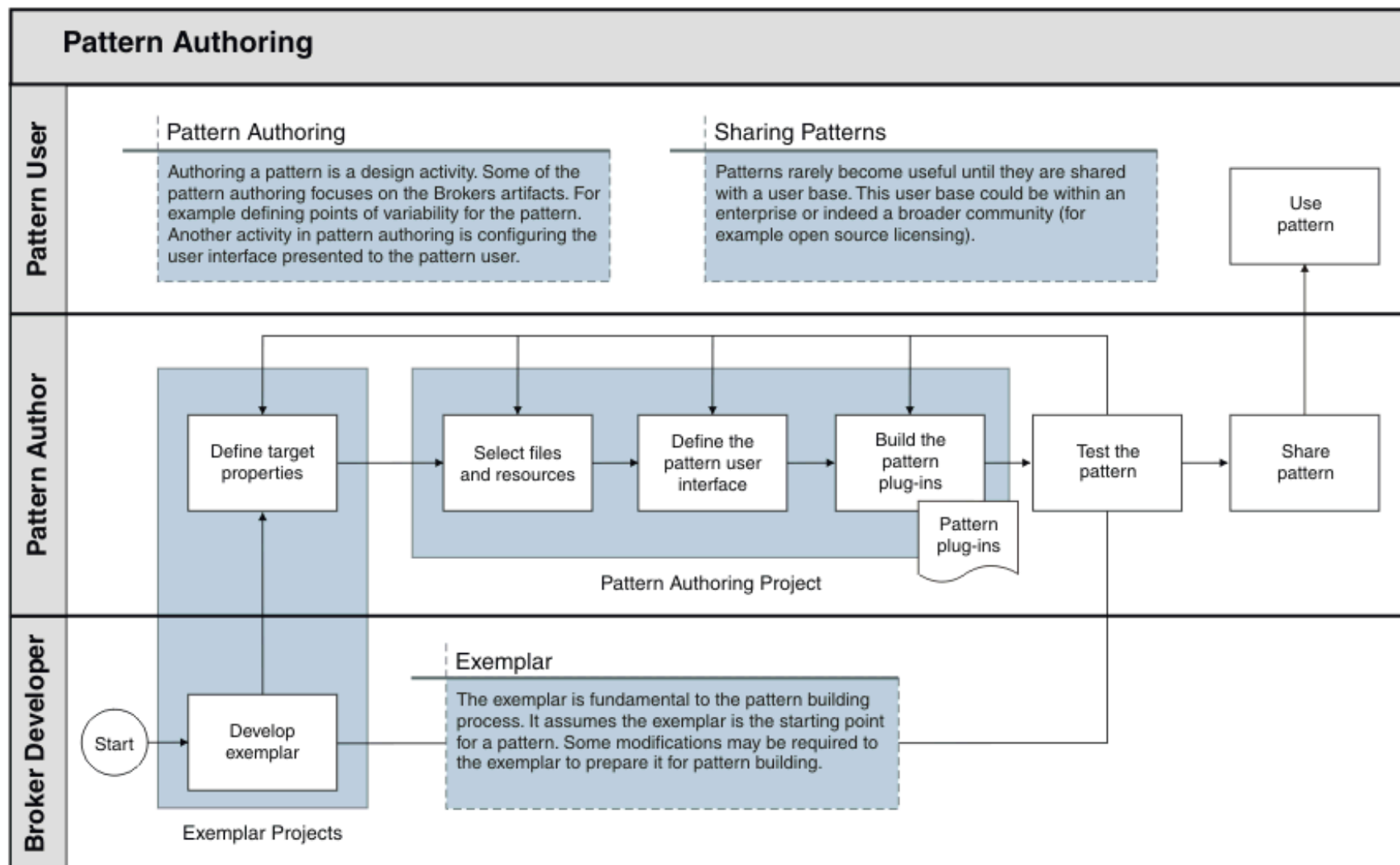
- Patterns and pattern authoring are first class concepts
- Pattern authoring is a design activity
 - It may be long lived
 - It is often not sequential
- Using patterns is a top-down activity driven by a requirement, but:
 - Authoring an exemplar is (typically) a bottom-up activity
 - So pattern authoring 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 into the exemplar projects to select target properties
 - Looks up to the user interface that will be presented to the pattern user
- Provide guidance so that a pattern author can create exemplars ready for pattern authoring

Workflow

Workflow



The Exemplar

The Exemplar

- An exemplar is a Broker solution to a problem – it includes Broker artefacts such as message flows and ESQL files. An exemplar acts as the start of a pattern. The pattern determines how the exemplar can be modified to generate a wider set of solutions.
- What are the starting points for an exemplar?
 - Recurring problem
 - A recurring business, or technical integration problem
 - Existing asset
 - A starting point for more general solutions
 - Architectural solution
 - An architecture where components can be realized through patterns
- The solution may require some re-design or modification to produce a good pattern exemplar. It should be remembered that the pattern, and its subsequent pattern instances, are only as good as the exemplar!

Variability in Patterns

Variability in Patterns (Part I)

Many kinds of variability are seen in the existing patterns!

- **Properties in message flows**
 - Modify property values at node or message flow level
 - Create or modify User Defined Properties (UDPs)

- **Structure of message flows**
 - Conditionally add or remove nodes and connections
 - Selection from alternative:
 - Subflow
 - Node type
 - Mapping / computation objects
 - Groups of nodes handling a specific capability
 - Handle repeating groups
 - Copy, configure and connect groups of nodes
 - Configure complex properties for routing

- **Computation**
 - Modify Java, PHP, ESQL using logic based on pattern parameters

Variability in Patterns (Part II)

■ Data Oriented

- Message sets
 - Create message sets
 - Modify message schemas
- Generation of schemas from pattern parameters and referenced data
 - Pattern parameters may include file names
 - Files may contain additional definitions
- Generation of transforms
 - By computation (Java, ESQL, PHP), XSLT or mapping nodes

■ Configuration Data

- Connectivity, for example WebSphere MQ scripts
- Database oriented – SQL scripts
- Other application oriented files
- Deployment instructions and scripts

■ Guidance and documentation

- Information and instructions using pattern parameters to tailor for each pattern instance

Pattern Parameters

Pattern Parameters (Part I)

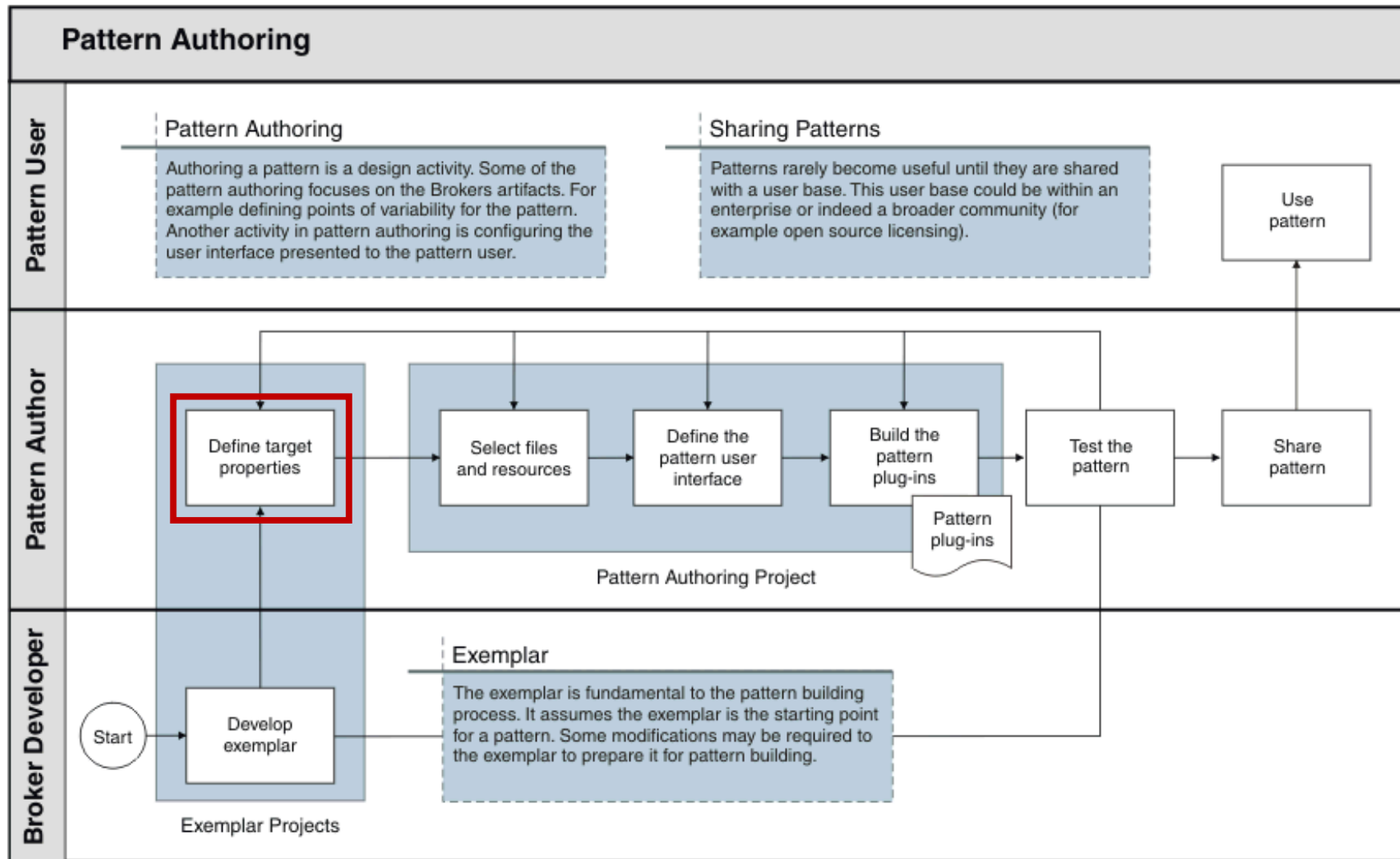
- Often sets one or more property values in a flow, UDP or node
- May constrain the value(s) of other pattern parameters
 - Message set, type and format is one good example
- May have dependencies that control what other parameters are enabled
 - Getting these dependencies correct has proven to be difficult
- Can be mandatory or optional
- Nearly always have a default value
- May constrain the permissible property values

Pattern Parameters (Part II)

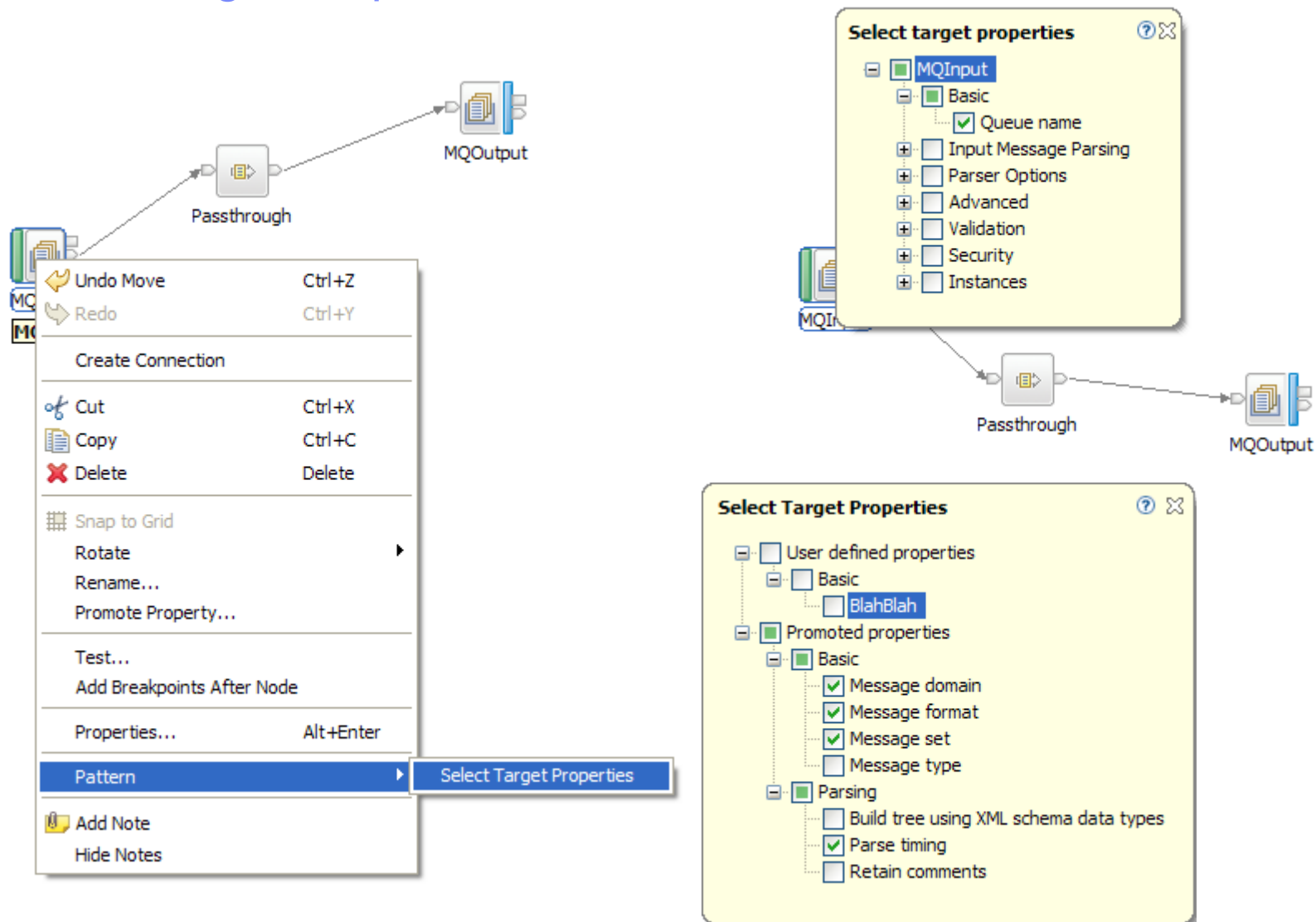
- May present the property values differently
 - For example a drop down list of enumerated values
 - Broker system default instead of an integer value
- May be used in logic statements to control the generation
 - For example, control the output of files, nodes or ESQL modules
- Are collected into display groups by the pattern author
 - Dependencies may be spread out across display groups
- Associated with an editor and optionally a validator
- All of the above behaviours can be combined...!

Design Walk Through

Define the Target Properties



Define the Target Properties



Design Points

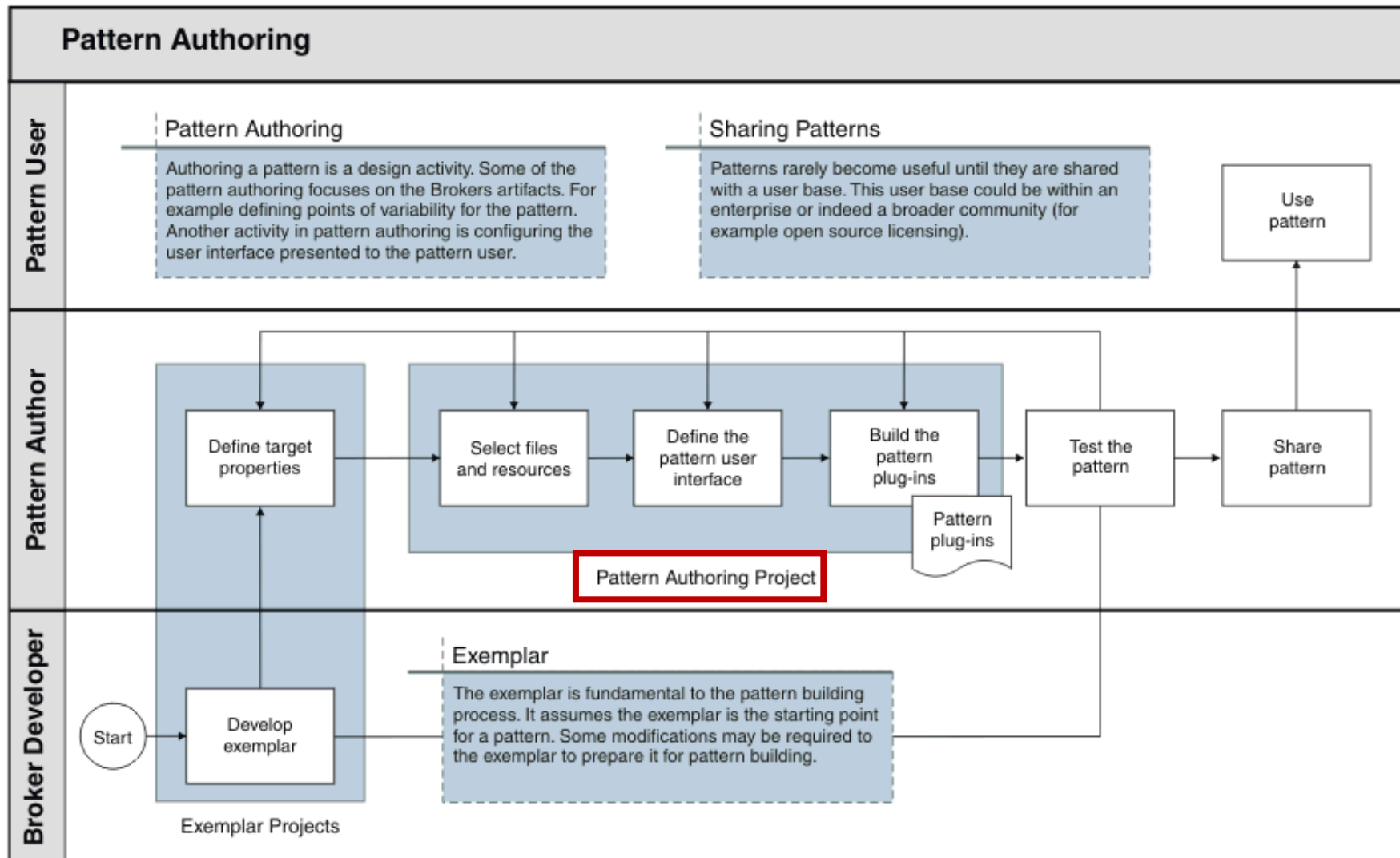
- Patterns are created in the Broker Toolkit
- Patterns are designed in a Pattern Authoring project and editor

Putting the pattern configuration in one of the exemplar projects was also considered. This approach was discarded because the choice of which project would have been arbitrary. Pattern Authoring projects are therefore slightly different from UDN projects - UDN projects contain the flows and the UDN configuration (such as `palette.xmi`).

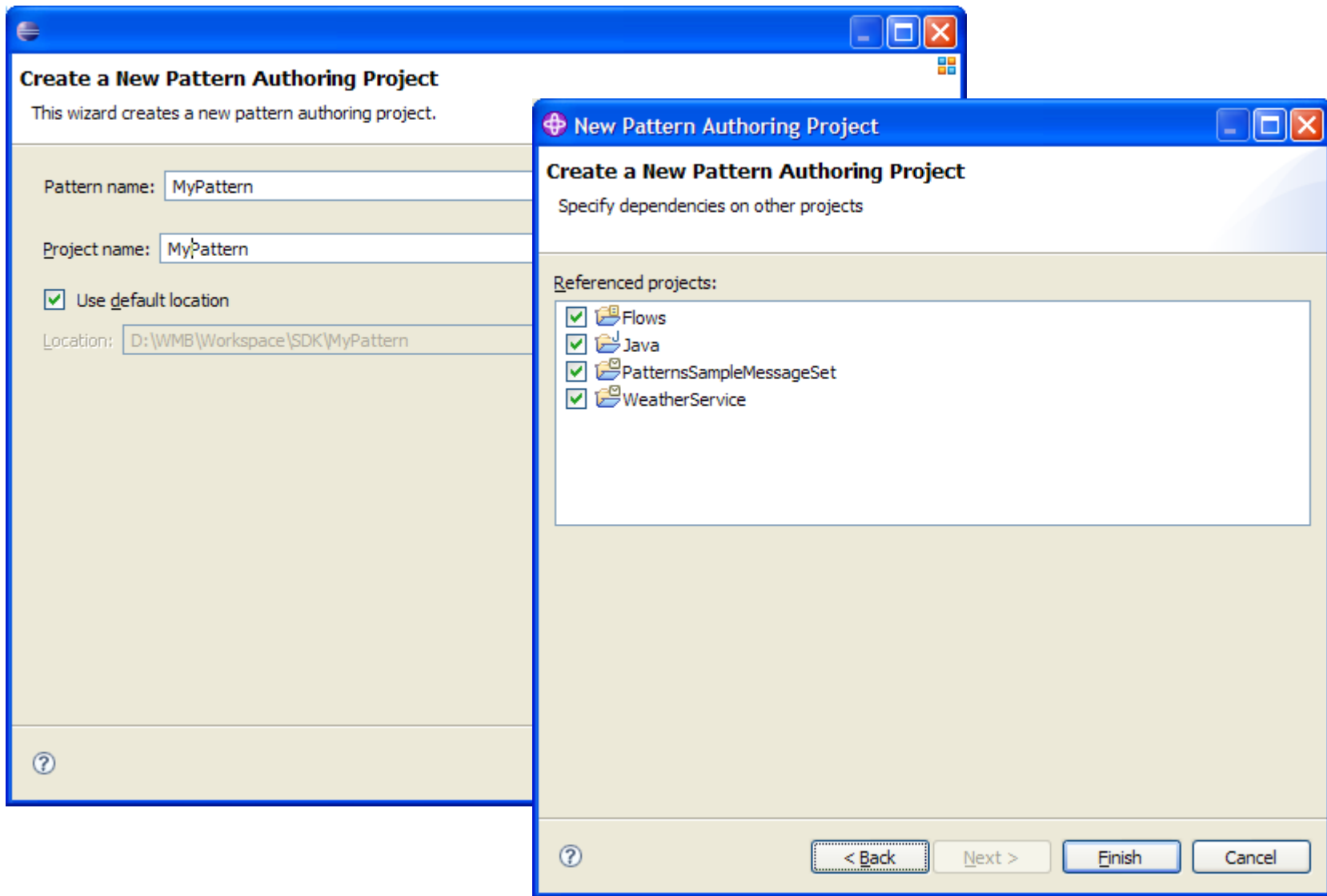
- Variability is expressed in the Toolkit editors
 - The first release supports property variability in the Flow Editor
 - Property variability is based on flow and User Defined Properties (UDPs)

The editors are the natural place for pattern authors to work (especially true in the future when variability includes making structural changes to flows). Adding variability in the editors also supports a use case where exemplar projects are used in multiple patterns.

Create A Pattern Authoring Project



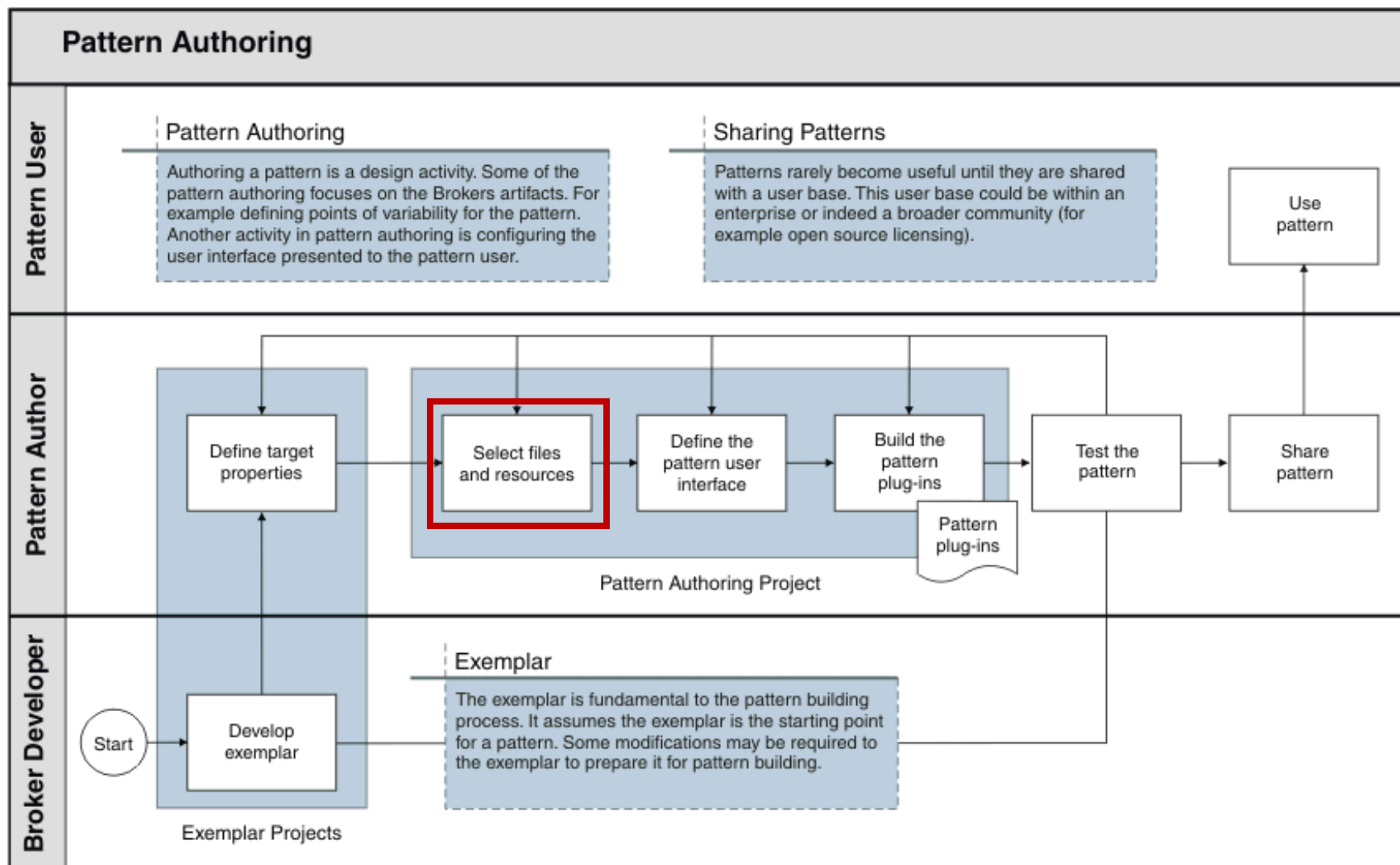
Create A Pattern Authoring Project



Design Points

- A pattern can generate one or more message flow projects
 - Support will follow for message sets, Java and other project types
- 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

Source Files

Select the source files to include in your pattern. You can also view the target properties available in those source files here.

Select Source Files

- Flows
 - mqsi
 - Request.esql
 - Request.msgflow
 - Response.esql
 - Response.msgflow
 - .project
 - Java
 - bin
 - JavaNode.class
 - src
 - JavaNode.java
 - .classpath
 - .project
 - WeatherService
 - Weather
 - com
 - cdyne
 - ws
 - weatherws
 - Weather.mxsd
 - Weather.wsdl
 - Weather_InlineSchema.mxsd
 - org
 - xmlsoap
 - schemas
 - soap
 - envelope

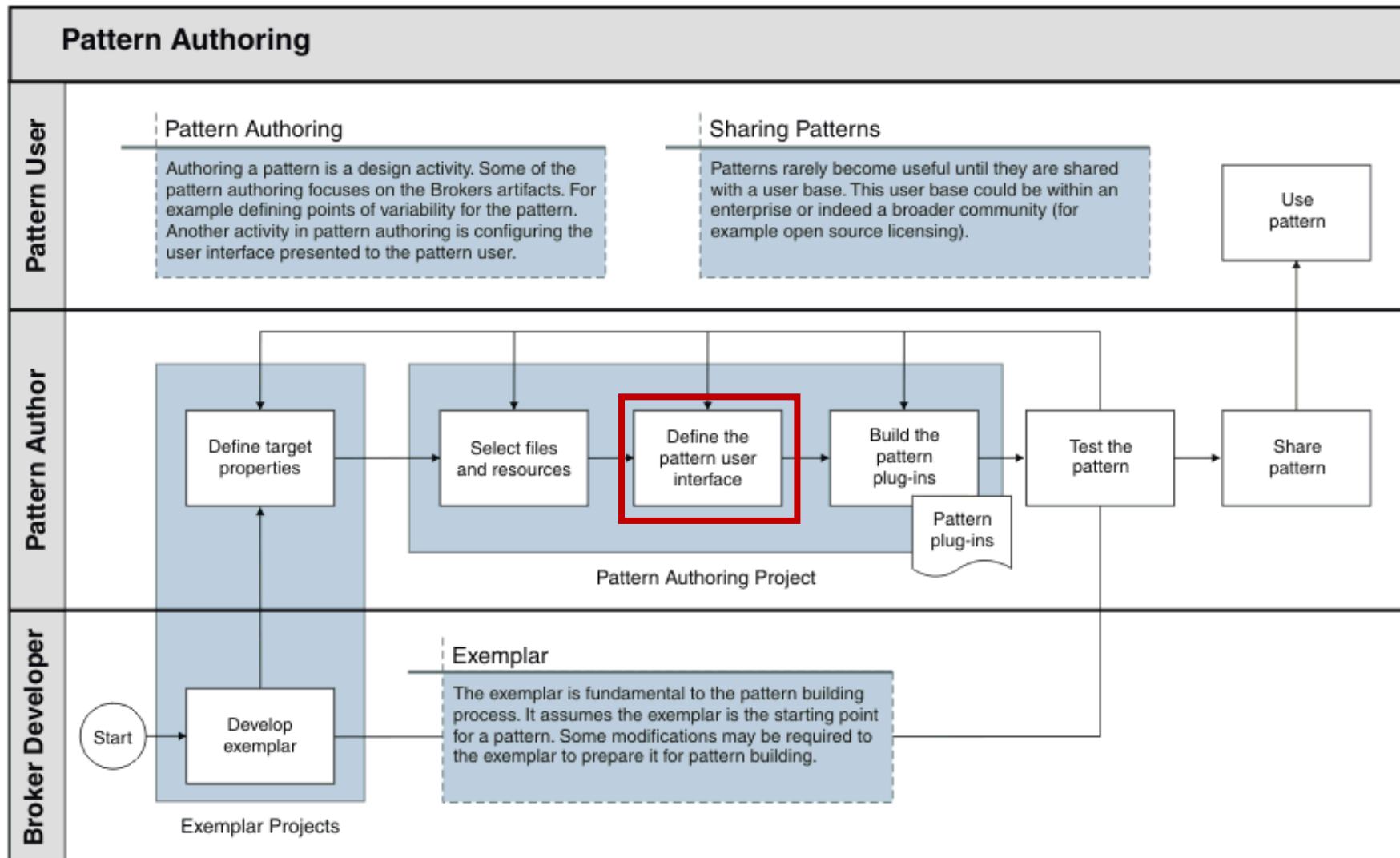
Target Properties

- Flows.mqsi.Request.MQInput.messageDomainProperty
- Flows.mqsi.Request.MQInput.queueName
- Flows.mqsi.Request.MQInput.validateTiming
- Flows.mqsi.Request.PromotedProperty.messageDomainProperty
- Flows.mqsi.Request.PromotedProperty.messageFormatProperty
- Flows.mqsi.Request.PromotedProperty.messageSetProperty
- Flows.mqsi.Request.PromotedProperty.validateTiming

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)

Pattern Parameters

Configure your pattern parameters and groups. Associate pattern parameters with target properties.

Groups and Parameters

- Default Group
 - Message Format Property (MQInput)
 - Flows.mqsi.Request.MQInput.messageFormatProperty
 - Message Set Property (MQInput)
 - Flows.mqsi.Request.MQInput.messageSetProperty
 - Message Type Property (MQInput)
 - Flows.mqsi.Request.MQInput.messageTypeProperty
 - Blah Blah (UserDefinedProperty)
 - Flows.mqsi.Request.UserDefinedProperty.BlahBlah
 - Queue Name (MQInput)
 - Flows.mqsi.Request.MQInput.queueName
 - Message Domain Property (MQInput)
 - Flows.mqsi.Request.MQInput.messageDomainProperty

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:

Default value:

Parameter editor:

Enumerated type:

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)

Pattern Parameters

Configure your pattern parameters and groups. Associate pattern parameters with target properties.

Groups and Parameters

- [-] [icon] General
 - [-] [icon] Logging
 - [icon] Flows.mqsi.Request.UserDefinedProperty.BlahBlah
 - [-] [icon] Queue name
 - [icon] Flows.mqsi.Request.MQInput.queueName
 - [-] [icon] **Input Information**
 - [-] [icon] Message set
 - [icon] Flows.mqsi.Request.MQInput.messageSetProperty
 - [-] [icon] Message domain
 - [icon] Flows.mqsi.Request.MQInput.messageDomainProperty
 - [-] [icon] Message type
 - [icon] Flows.mqsi.Request.MQInput.messageTypeProperty
 - [-] [icon] Message format
 - [icon] Flows.mqsi.Request.MQInput.messageFormatProperty

Add Group...

Add Parameter...

Edit...

Delete

Enumerated Types...

[icon]

[icon]

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

Categories

Create new categories and assign your pattern to a category.

Categories

The screenshot displays the 'Categories' configuration window in IBM Integration Designer. On the left, a tree view shows the following structure:

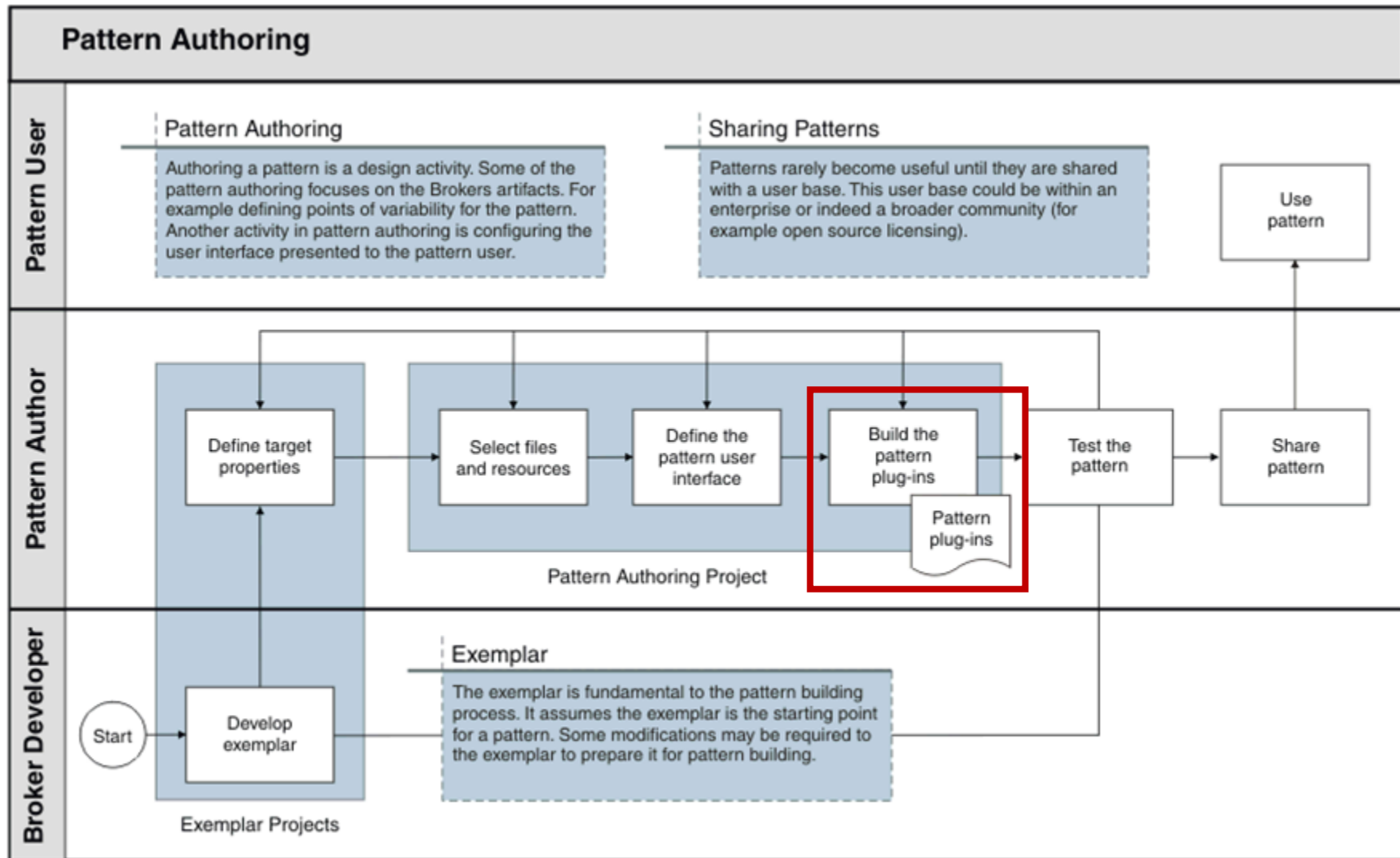
- Patterns
 - Service Virtualization
 - Service Proxy
 - MyPattern
 - File Processing
 - Record Distribution
 - Application Integration
 - SAP
 - Message-based Integration
 - Message Splitter
 - Message Correlator
 - Service Enablement
 - Service Facade
 - Service Access

On the right side of the window, there are three buttons: 'Add Category', 'Remove Category', and 'Edit Category'. At the bottom right corner, there are two icons: one for adding a category (upward arrow) and one for removing a category (downward arrow).

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

Create Pattern

Test your pattern by configuring your pattern plug-in information, click "Create Pattern Plug-ins", and click Test Pattern".

Plug-in Information

Configure the unique identifier for your pattern plug-in.

Pattern name:	<input type="text" value="MyPattern"/>	
Plug-in ID:	<input type="text" value="com.your.company.domain.MyPattern"/>	
Version:	<input type="text" value="1.0.0.0"/>	
Provider:	<input type="text" value="Your Company Name"/>	
Description:	<input type="text" value="Plug-in created by the Pattern Authoring editor"/>	

Translation Options

If you enable this option, the Pattern Authoring editor creates two additional NLS plug-ins. These plug-ins are set up so that you can drop in translated resources, such as Java property files. If you are creating a single language pattern, do not select this check box.

Create translation plug-ins (*.nl1 and *.doc.nl1)

Pattern Distribution

After you have created and tested your pattern plug-ins (see the Plug-in ID above for the plug-in names), package your pattern by clicking "File > Export > General > Archive File" to export these plug-ins.

To use the pattern plug-ins, the pattern user must extract the exported archive files into the default WebSphere Message Broker Toolkit plug-ins directory:

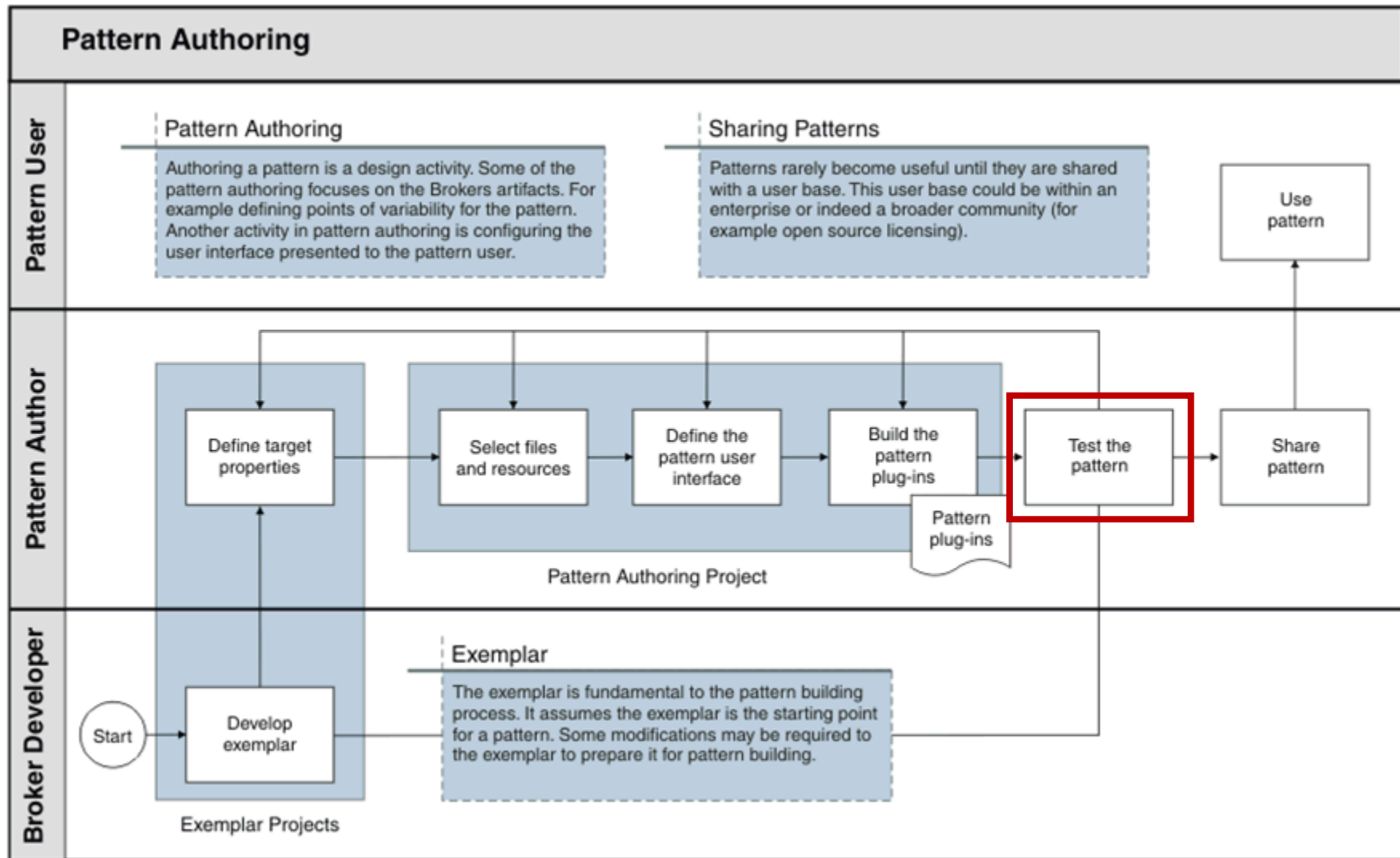
Design Points

- The Pattern Authoring Editor creates Eclipse plug-ins
 - Eclipse provides excellent support for plug-ins and features for distribution

The Pattern Authoring Editor could also generate a feature plug-in along with the pattern plug-ins. However Eclipse has good support for creating feature plug-ins. Rather than duplicate existing Eclipse functionality, the Pattern Authoring Editor will provide the pattern author with a short cut to start the wizard.

- 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

The screenshot shows the IBM WebSphere Patterns Explorer interface. On the left, a tree view displays various pattern categories under 'Patterns', including 'Application Integration', 'File Processing', 'Message-based Integration', 'Service Enablement', and 'Service Virtualization'. The 'My Pattern' is highlighted in the tree. On the right, the 'View Pattern Specification' window is open, displaying the following content:

View Pattern Specification

View information about the selected pattern and click the "Create New Instance" button or click [here](#) to start using a pattern.

My Pattern

Solution

Related tasks

Read the following section for information about how to apply and use this pattern.

[Constraints on the use of the pattern](#)

[Tasks to complete before applying the pattern](#)

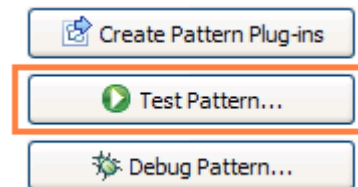
[Apply and configure the pattern](#)

[Parameters for the pattern](#)

[Tasks to complete after generating the pattern](#)

Design Points

- Testing a pattern requires a re-launch of the Toolkit
 - This could be improved through a simulation mode (similar to UDNs)
 - 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!

Expressions

Expressions

- There are many places in patterns where conditional logic applies:
 - Add this file to a pattern instance *if this condition* is true
 - Remove these nodes from a message flow *if this condition* is true
 - Add this project to a pattern instance *if this condition* is true
 - Enable this field in the pattern instance editor *if this condition* is true
 - ...
- The Pattern Authoring editor uses XPath as its expression language
 - XPath is a general purpose expression language!
 - XPath is normally used in conjunction with an XML DOM tree
 - In pattern authoring there is no XML DOM tree, just the core language
 - XPath 1.0 is currently supported (XPath 2.0 in due course)
- 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
 - Each XPath expression is evaluated and the result assigned to the parameter

Edit Parameter Queue Name (MQInput)

Configure the Pattern Parameter

Configure the pattern parameter and how it is displayed to pattern users.

Basic Editor **Expression** Enablement

Configure an XPath expression that transforms the value of this parameter. The XPath expression is evaluated when a pattern instance is created. The XPath expression can get the value of other parameters using the `getValue()` XPath function.

Functions

- String
 - string
 - concat
 - starts-with
 - contains
 - substring-before
 - substring-before
 - substring

Operators

- +
-
- *
- div
- =
- !=
- <
- <=
- >

Function name: Operator:

Pattern Parameters

Groups and Parameters	Parameter ID	Test Value
<ul style="list-style-type: none"> UserDefinedProperty <ul style="list-style-type: none"> Logging (UserDefinedProperty) Queue Name (MQInput) MQInput <ul style="list-style-type: none"> Message Format Property (MQInput) Message Set Property (MQInput) Message Type Property (MQInput) Message Domain Property (MQInput) 	pp1 pp5 pp2 pp3 pp4 pp6	MRM

Test value: Parameter ID:

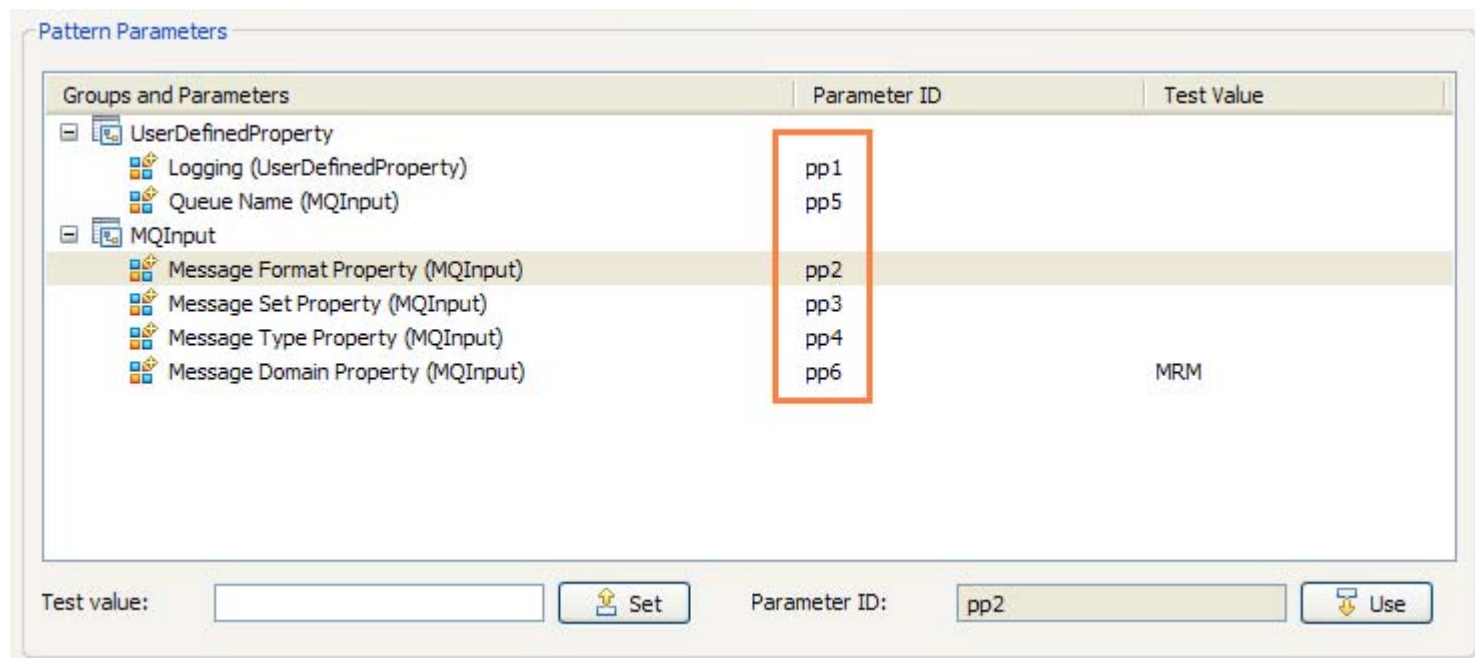
Expression Evaluation

Expression:

Result:

Design Points

- Every pattern parameter is assigned a unique immutable ID
- Parameter IDs follow a simple naming convention: pp1 . . . n



- The Pattern Authoring editor adds an XPath function called `getValue`
 - `getValue()` returns the value of any pattern parameter given a parameter ID

Enablement

Configure Pattern Parameters

Provide values for pattern parameters. Click the "Generate" button or click [here](#) to generate a pattern instance.

i Pattern parameters are ready. Click the "Generate" button to generate a pattern instance.

Pattern Parameters

▼ Input information ✓

Input source and validation requirements

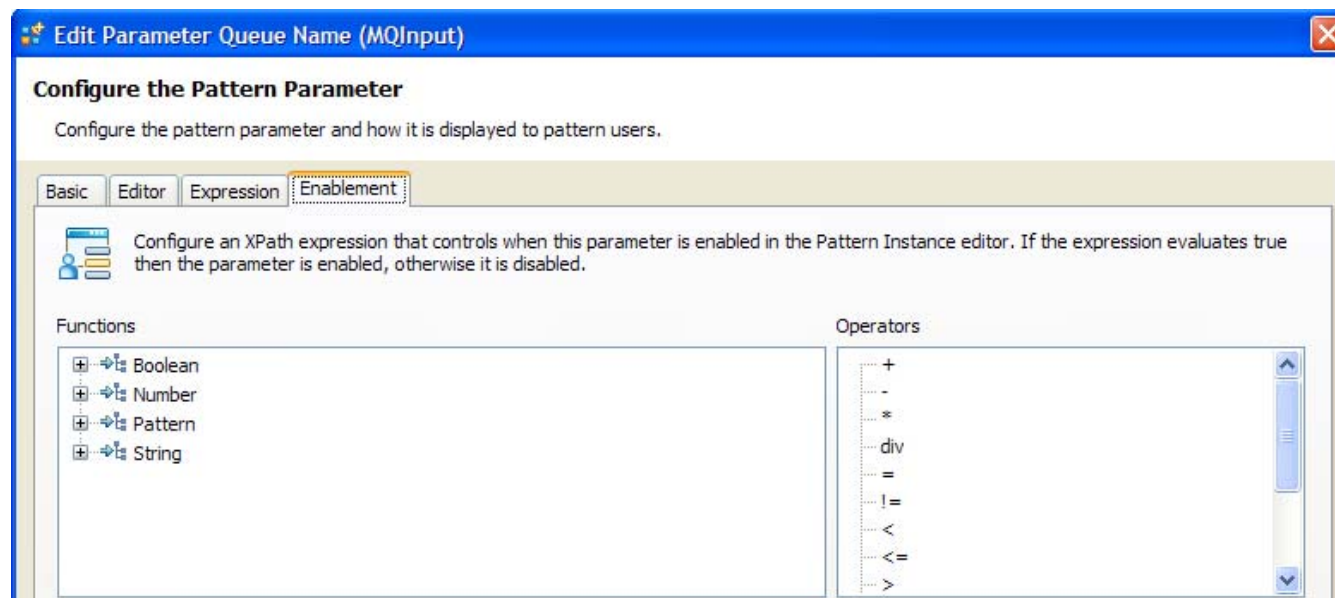
Input queue *	IN
Validation of input messages	None ▼
Input message set *	▼
Input message type *	▼
Input message format *	▼

▶ Provider information ✓

▶ Response information ✓

Enablement

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

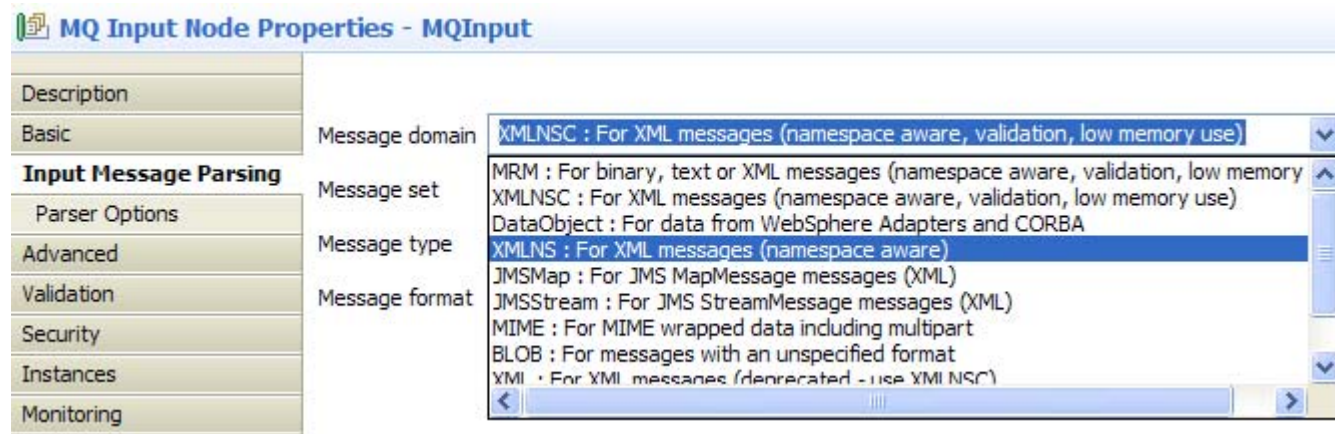


- The enablement expression is evaluated every time a pattern parameter referenced in the XPath expression changes value
- If the result of the evaluation is true then the editor is enabled

Enumerated Types

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

Configure Enumerated Types
✕

Configure Enumerated Types

Configure the enumerated types used by pattern parameters.

An enumerated type is a set of values which a pattern parameter will accept. The display names are presented in a drop down list to pattern users when they create an instance of this pattern. The value is the actual content stored in the target property.

i This enumerated type is defined by a target property and cannot be deleted

Enumerated type: Enumerated type for Flows.mqsi.Request.MQInput.messageDomainProperty ▼

Add
Remove
Rename

Display Name	Value	
MIRM : For binary, text or XML messages (namespace aware, validation, low memory use)	MIRM	Add
XMLNSC : For XML messages (namespace aware, validation, low memory use)	XMLNSC	
DataObject : For data from WebSphere Adapters and CORBA	DataObject	Remove
XMLNS : For XML messages (namespace aware)	XMLNS	
JMSMap : For JMS MapMessage messages (XML)	JMSMap	
JMSStream : For JMS StreamMessage messages (XML)	JMSStream	
MIME : For MIME wrapped data including multipart	MIME	
BLOB : For messages with an unspecified format	BLOB	
XML : For XML messages (deprecated - use XMLNSC)	XML	

Reset Values

You cannot remove an enumerated type if it is being used by a parameter or is defined by a target property. You can change the type of a pattern parameter using the Edit Parameter button in the Pattern Parameters tab.

Parameters using this enumerated type:

- Input Information
 - Message domain
 - Flows.mqsi.Request.MQInput.messageDomainProperty

OK
Cancel

Questions and feedback!

