

ChainBuilder ESB

Visual Enterprise Integration™

Version 1.0

Component Flow Editor Guide



©Copyright 2007
Bostech Corporation
2800 Corporate Exchange Drive
Suite 260
Columbus, OH 43231

Acknowledgements

This document contains proprietary information that is the property of Bostech Corporation. Any reproduction, disclosure, or transfer of this document or the information contained herein without the express written consent of Bostech Corporation is strictly prohibited.

The use of the information contained in this document and the implementation of any of its techniques are the sole responsibility of the client and depend on the client's ability to evaluate the information and implement it into the client's operational environment.

Except for any express written warranties made by it, Bostech Corporation makes no warranties or representations with respect to any information contained herein, whether express, implied, statutory, or otherwise, in fact or in law, including without limitation, any implied warranties of merchantability or fitness for a particular purpose; and in no event shall Bostech Corporation be liable for any special, consequential, indirect, punitive, or exemplary damages in connection with the use of the information contained herein. The information contained in this document is subject to change at any time without notice.

Trademarks

The following trademarks and acknowledgments apply to the information presented in this manual:

- ChainBuilder is a registered trademark of Bostech Corporation.
- Adobe and Acrobat Reader are registered trademarks of Adobe, Inc.
- Java is a registered trademark of Sun Microsystems, Inc.
- Windows (NT, 2000, XP, and Server 2003), .NET Framework, Internet Information Services (IIS) are registered trademarks of Microsoft Corporation.

Credits

The following third-party products are used within the ChainBuilder product, and acknowledgments apply to the information presented in this manual:

- Acrobat Reader is created and licensed by Adobe, Inc.
- This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)
- This product includes software developed by Eclipse (<http://www.eclipse.org/>)

Table of Contents

1. Introduction.....	1
1.1. JBI Terminology Overview	1
1.2. Component Flow Editor Overview	2
1.2.1. Component Palette.....	2
1.2.2. Canvas	3
1.2.3. Properties Panel.....	3
2. Starting the Component Flow Editor.....	4
3. Adding Components to the Canvas.....	5
3.1. Binding Components.....	5
3.1.1. HTTP Component.....	5
3.1.2. File Component.....	5
3.1.3. FTP Component.....	5
3.1.4. JMS Component	5
3.2. Service Engines.....	5
3.2.1. Transformer Service Engine	6
3.2.2. Parser Service Engine	6
3.2.3. XSLT Service Engine.....	6
3.2.4. Sequencer Service Engine	6
3.2.5. Context-Based Router (CBR) Service Engine.....	6
3.2.6. Script Service Engine	7
3.2.7. JDBC Service Engine.....	7
3.3. External Systems	7
3.4. Connections	7
4. Adding Custom Code.....	9
4.1. Creating Custom Code Files.....	9
4.2. Using Custom Code.....	14
4.2.1. User Points of Control (UPOC)	14
4.2.2. Transaction ID (TrxId).....	16
4.2.3. Script Component	18
5. Detailed Component Descriptions	18
5.1. HTTP Binding Component.....	19
5.2. File Binding Component.....	19
5.3. FTP Binding Component	22
5.4. JMS Binding Component.....	25
5.5. Transformer Service Engine	27
5.6. Parser Service Engine	27
5.7. XSLT Service Engine	28
5.8. Sequencer Service Engine	28
5.9. Content-Based Router (CBR) Service Engine	28
5.10. Script Service Engine.....	29
5.11. JDBC Service Engine	30
6. Generating the Service Assembly Archive.....	31
7. ChainBuilder ESB Community.....	34

1. Introduction

1.1. JBI Terminology Overview

The ChainBuilder ESB Component Flow Editor provides an easy to use graphical method for creating JBI Service Assemblies. Since the Component Flow Editor allows the creation of JBI Service Assemblies, it is necessary to have a basic understanding of the JBI standard and some of the objects it defines.

Binding Component - There are two types of components in the JBI specification, the first is a Binding Component. A binding component provides an "endpoint" to the outside world. The job of a Binding Component is to convert between the protocol-specific data of an external system and the normalized message format used internally by all JBI components.

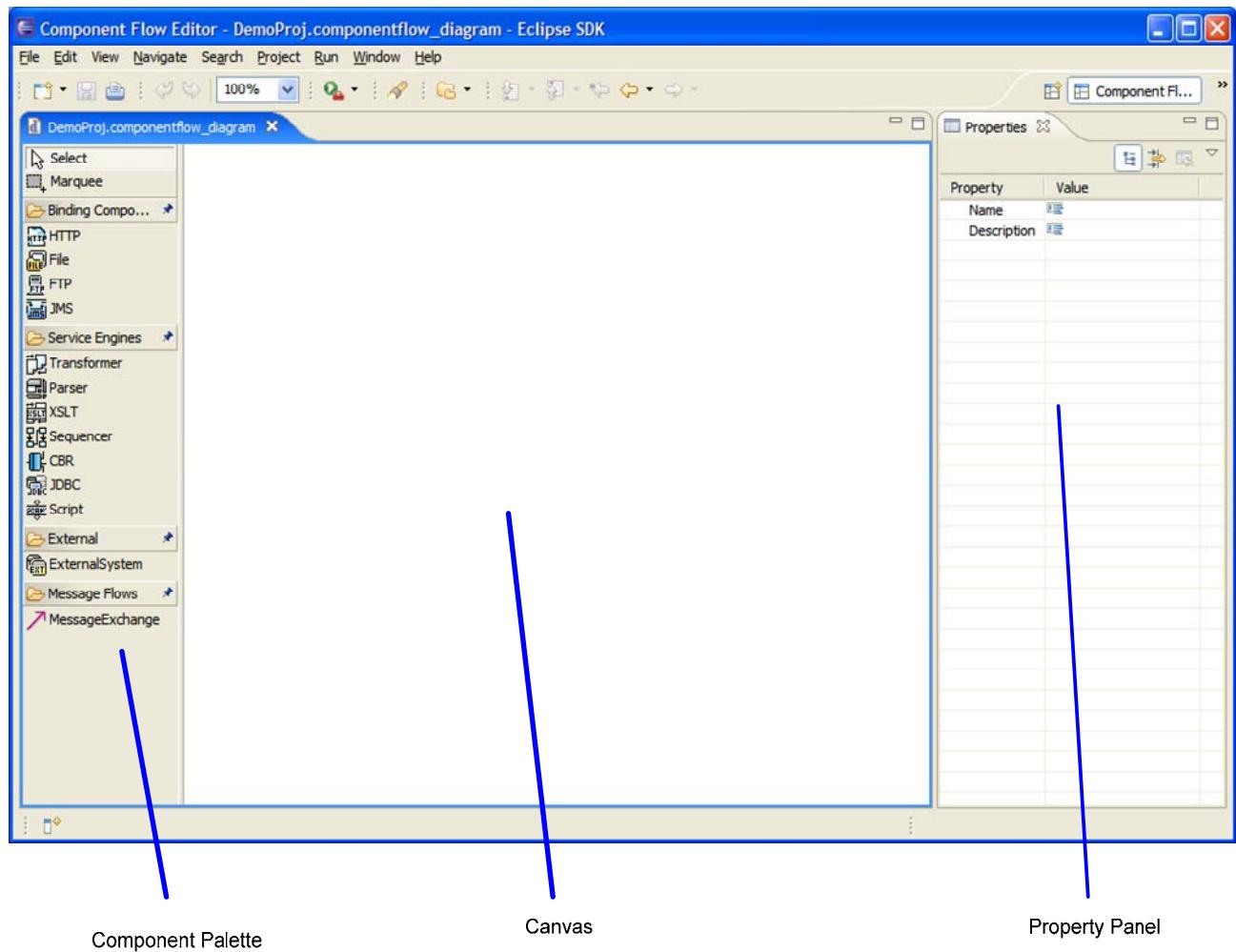
Service Engine - The other type of component is the Service Engine. Service Engines provide the business logic of a JBI solution. This includes functionality like data transformation, business process orchestration, message routing and many other processes. Service Engines use JBI Normalized Messages as their input and output, so to communicate with a Service Engine from the outside world, a Binding Component must first receive the data and convert it to a Normalized Message, then it can be sent to the Service Engine.

Service Unit - A Service Unit is a package containing all of configuration settings for a single component instance. This package includes the component specific settings as well as any "artifacts" that are used by that component. An example of a Service Unit artifact is an XSL style sheet used by an XSLT processor Service Engine.

Service Assembly - A Service Assembly is a package containing one or more Service Unit packages and usually information about the interconnections between those Service Units.

1.2. Component Flow Editor Overview

The Component Flow Editor contains three main parts, the Component Palette, the Canvas and the Properties Panel. Components that are to be added to a Service Assembly are selected from the Palette and placed on the Canvas. Then each component on the Canvas is configured using the Properties Panel.



1.2.1. Component Palette

The Palette is divided into different categories. The first two items are the selection tools, "Select" and "Marquee". The Select tool is the default cursor which allows the manipulation of individual items. The Marquee tool allows multiple items to be selected on the canvas. This can be a convenient way to move or delete a group of components.

The Binding Component and Service Engines categories contain all of the available JBI components that may be used to build a Service Assembly.

The External category contains a single item "External System". This is to represent an external entity that connects to a Binding Component. This is purely for documentation purposes to help give a clear picture of what is communicating with the Service Assembly.

The last category is Message Flows. This provides the tool to connect the different Service Units on the Canvas to direct the flow of data.

1.2.2. Canvas

The Canvas area is where the Service Assembly is built. Components and connections are drawn on the canvas to create the Service Assembly. When the Service Assembly is complete, the Service Assembly archive package that is used by the JBI Container must be created. This can also be done using the Canvas area by right clicking on it to open the context menu and selecting the "Deploy" option.

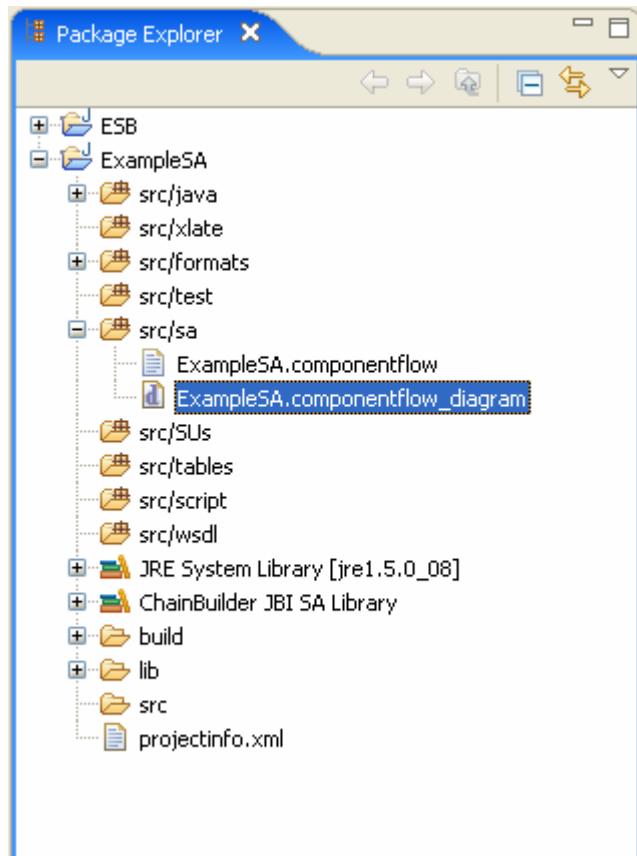
1.2.3. Properties Panel

The properties panel displays the settings for each item in the Service Assembly. The settings available in the properties panel depend on which type of object is selected in the Canvas. For example, the settings for a File Binding Component will be different from the settings available for a Transformer Service Engine.

2. Starting the Component Flow Editor

To start the Component Flow Editor, create a new JBI Service Assembly project or open an existing JBI Service Assembly project. In the `src/sa` folder of the project are two files.

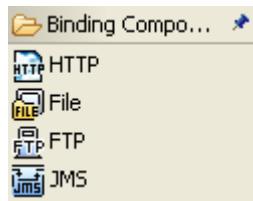
Double click on the `{ProjName}.componentflow_diagram` file, this will start the Component Flow Editor.



3. Adding Components to the Canvas

To build a Service Assembly, components are added to the Canvas from the Component Palette.

3.1. Binding Components



To add a Binding Component to the Canvas, click the desired Binding Component on the palette to select it as the current item. Then click again on the Canvas at the location where the component should be added. When a Binding Component is added to the Canvas, it represents a Service Unit. The name of the Service Unit may be edited once it is added.

3.1.1. HTTP Component

The HTTP Component is used to provide an HTTP server for external HTTP clients to connect to or provide an HTTP client that can retrieve information from external HTTP Servers. It can be used to provide or invoke web services over HTTP.

3.1.2. File Component

The File Component is used to read or write data on the local file system. It is able to process XML as well as non-XML data. It also provides automatic archiving functionality and also the ability to determine if a file is still being written to by an external system, so it is not processed before all of the data is present.

3.1.3. FTP Component

The FTP Component is used to send or receive files using the FTP protocol. It is able to process XML as well as non-XML data. It also provides automatic archiving functionality and also the ability to determine if a file is still being written to by an external system, so it is not processed before all of the data is present.

The FTP component can also be used in the scripting mode which allow the FTP component to receive the XML based scripting language to perform advanced FTP operations.

3.1.4. JMS Component

The JMS Component is used to read or write data using a JMS compliant Queue based messaging system such as ActiveMQ or WebSphere MQ Series.

3.2. Service Engines



A Service Engine is added to the Canvas exactly the same as a Binding Component. Click the desired Service Engine on the palette to select it as the current item. Then click again on the Canvas at the location

where the component should be added. When a Service Engine is added to the Canvas, it represents a Service Unit. The name of the Service Unit may be edited once it is added.

3.2.1. Transformer Service Engine

The Transformer Service Engine uses mappings created using the ChainBuilder ESB Map Editor to convert between messages between different formats. This allows conversions between XML messages, custom formats based on fixed length and variable length records, as well as standards based messages like X12 EDI formats.

3.2.2. Parser Service Engine

The Parser Service Engine provides the ability to convert non-XML messages to an XML representation which can then be processed by XML specific components such as the XSLT Service Engine. The non-XML messages can be standards based like X12 or defined using the ChainBuilder ESB Message Format Editor.

3.2.3. XSLT Service Engine

The XSLT Service Engine transforms XML data contained in a Normalized Message using standard XSL style sheets. This can be used to generate different XML, HTML and many other types of documents from XML data.

3.2.4. Sequencer Service Engine

The Sequencer Service Engine is used to chain together multiple service engines or binding components. The normal operations of JBI components are to act as a consumer and provider. This means that a request is sent from one component to another component and then a response may optionally be returned. To be able to forward the results of the service provider to another component besides the original consumer, the Sequencer component must be used. It uses a list of services to take the results from one service to be used as the input for the next service in the list. The results of the last service in the list can then be sent back to the original consumer component.

3.2.5. Context-Based Router (CBR) Service Engine

The CBR Service Engine is used to dynamically route message into different destination endpoints based on message content. You can specify the Transaction Identification (TrxId) in the CBR component. When a request is sent from a source component to a CBR, the CBR will perform TrxId operation on the incoming request and return a string value to identify the request. The CBR will then route the request to different destination based on the routing rules defined in CBR component. The supported TrxID types are Fixed, CSV, X12, Script and XPath. The routing rules can be based on XPath, exact matching and regular expression based matching.

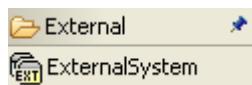
3.2.6. Script Service Engine

The Script Service Engine provides the ability to write your own custom logic not provided out-of-the-box by ChainBuilder ESB. You can use the Groovy or POJO (Plain Old Java Object) as scripting choice. The Script component can act as a consumer or provider. The ideal use of Script component in a provider role is to implement your own business logic to incoming request. The ideal use of Script component in a consumer role ranges from writing special file processing to creating your own socket server to perform HL7 minimum lower layer protocol (MLP).

3.2.7. JDBC Service Engine

The JDBC Service Engine accepts the XML-based request messages which contain JDBC compliant SQL statements. It will execute the SQL statement and return the response message which contains information about the state of the request as well as possible row results.

3.3. External Systems



The External System component simply document external connections to the JBI Service Assembly. The External System does not perform any function at runtime and is not actually packaged as part of the Service Assembly. Its purpose is to give a clear picture of what external system is connecting to a particular Binding Component Service Assembly.

An External System object is added to the Canvas in the same manner as Binding Components and Service Engines. To add an External System to the Canvas, select it as the current object in the Palette, and then click on the Canvas at the location where the External System should be added. The name of the external system can then be edited.

3.4. Connections



The Message Exchange tool is used to connect different Service Units and External Systems on the Canvas. To add a connection, select the Message Exchange object in the Palette. Then click on the Service Unit on the Canvas that acts as the Consumer in the connection. Then click on the Service Unit that acts as the Service Provider in the connection. An arrow will be drawn between the two Service Units.

The Editor will select the correct style of arrow to use based on the components that are being connected. The different types of connections are:

External System Connections - This type of connection is used to show a connection between an External System and a Binding Component. The color of the arrow will be blue, which is the same color as the External System component.

Standard Message Exchange - This type of connection is used to show a connection between a two components. The components may be Binding Components or Service Engines. The color of the arrow will be black for this type of connection.

Sequencing Message Exchange - This type of connection is used to show the Service Units that a Sequencing Component will invoke. The color of the arrows for this type of connection is yellow. Each arrow also will have a number on it to indicate the order that the Service Units will be called.

Context-Based Router (CBR) Message Exchange - This type of connection is used to show the connection from a CBR component to target components. The color of the arrows for this type of connection is yellow. You can define the matching type and expression in each connection. Each arrow may optionally display an expression if the matching type is “Exact” type.

The arrow also displays the MEP (Message Exchange Pattern) that will be used by using different style arrowheads. For “In Only” defaultMEP, the arrow will have a normal arrowhead pointing from the consumer to the provider. For “In Out” defaultMEP, the arrow will have a diamond arrowhead at the provider end of the connection.

4. Adding Custom Code

User defined classes or scripts can be used in a variety of places.

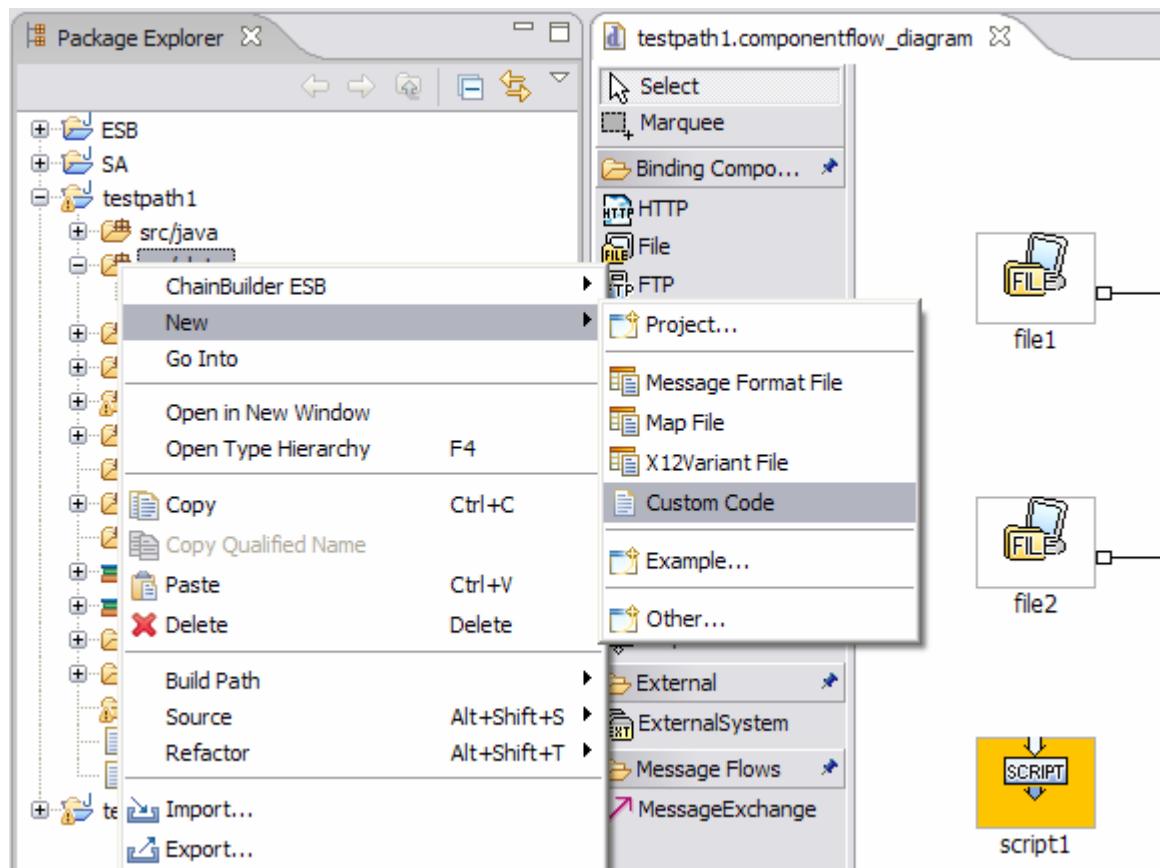
- UPOC (User Points of Control). These are scripts that can operate on a message exchange as it flows between components. These can be Java classes or Groovy script.
- User Defined Mapping Operation. These can be user operations of filter methods that help transform data in a map. Currently, these can only be Java classes. Groovy script will be added later.
- TrxId (Transaction ID). This is an identifier used by the Content Based Router. It can use a Java class or Groovy Script.
- Script component. This can be either a Java class or Groovy Script. This allows you to create an entirely new component with your own code.

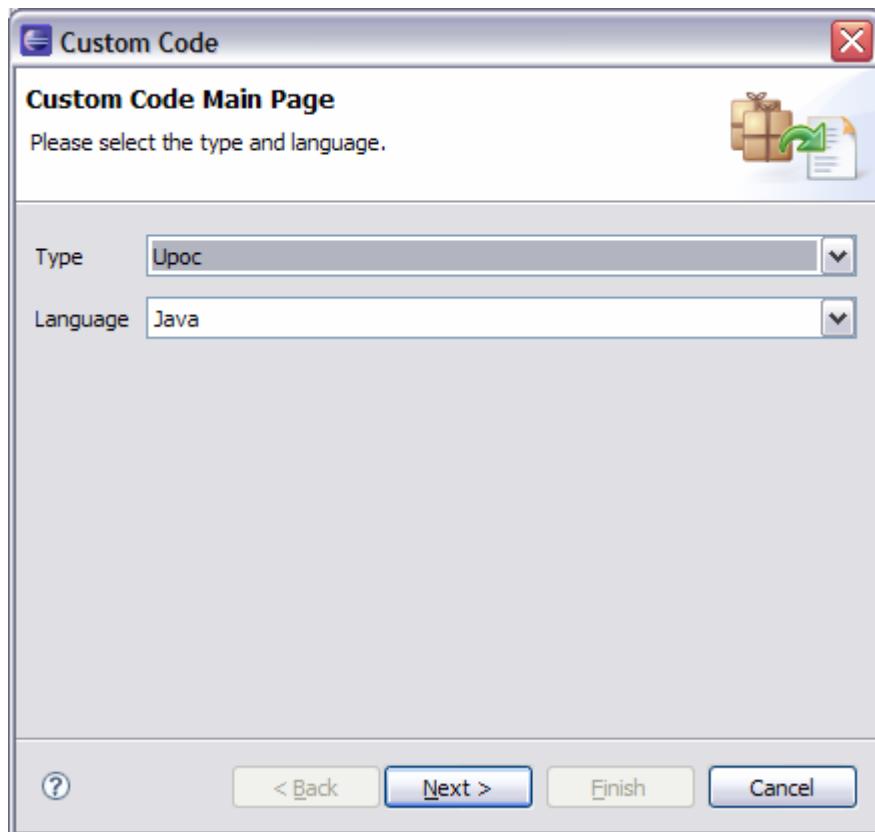
User defined code usually implements a specific interface and follows a basic pattern. The IDE has the ability to create skeleton files that make it easy for you to create custom code.

4.1. Creating Custom Code Files

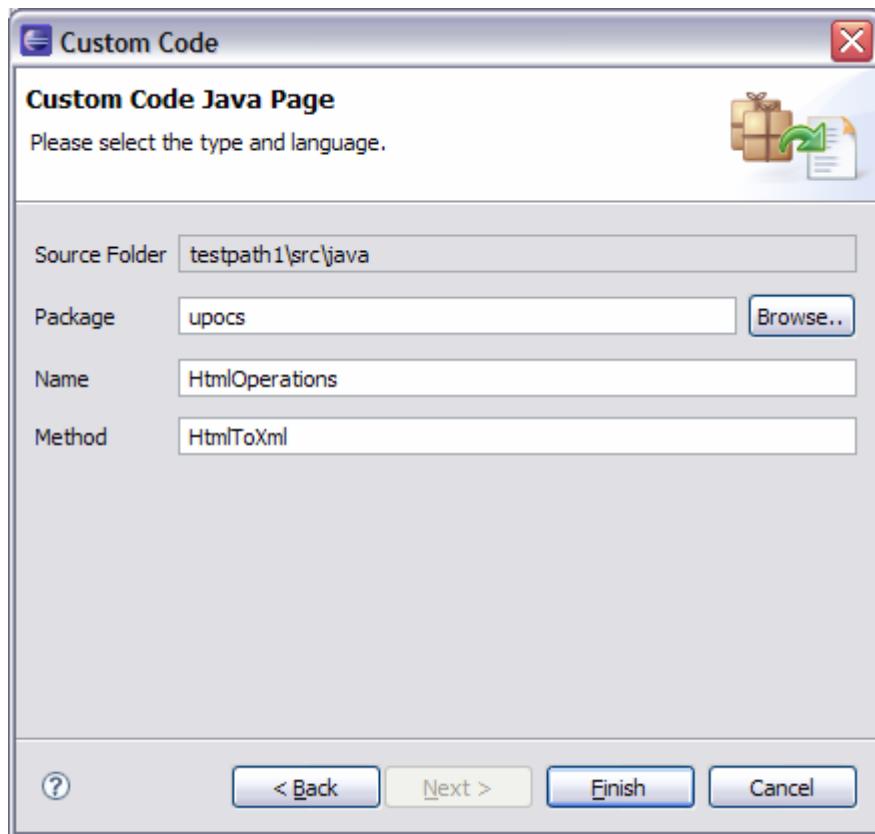
If you will be using Java for your custom code then you must first create a package for your classes to reside in. You can name the packages any way you like and nest them. To create a package, right-click on src/java in the package explorer. Then select “New→Other→Java→Package”. Enter a package name.

To create custom code skeletons, right-click anywhere inside of the project in the package explorer, then select “New→Custom Code”. This starts the Custom Code Wizard.

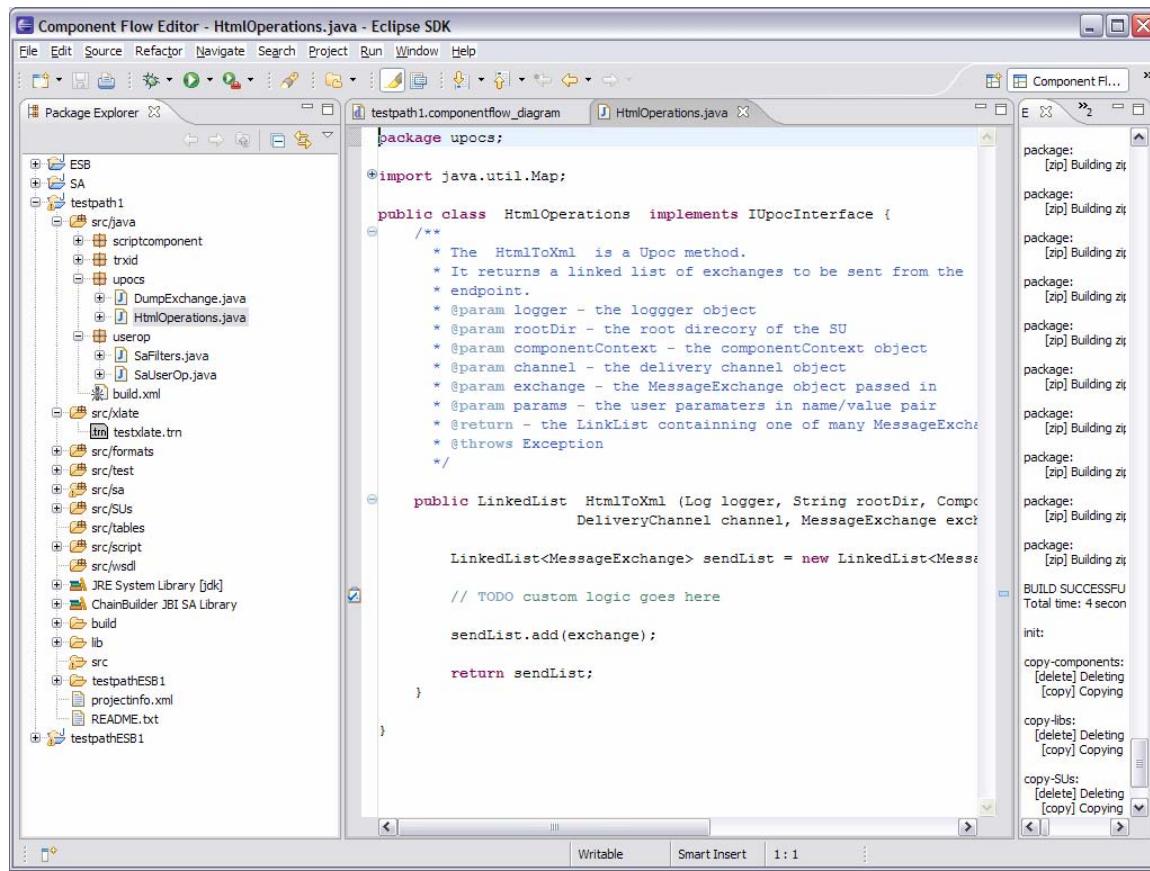




Select the Language first and then the Type. Language is at the bottom but you must select it first. When you click “Next”, you are given options appropriate to the type of code and Language selected. This shows the screen for a Java UPOC.

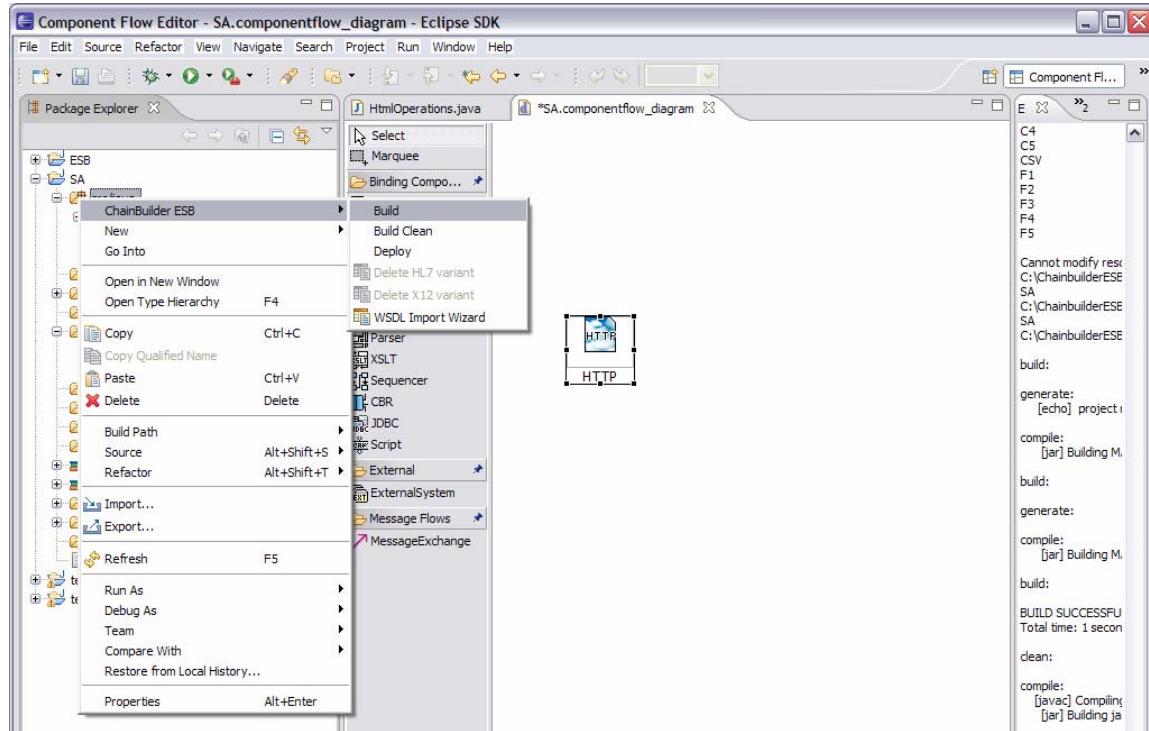


When you click “Finish”, a skeleton file is created for you. You can open this file and add your custom logic to it.



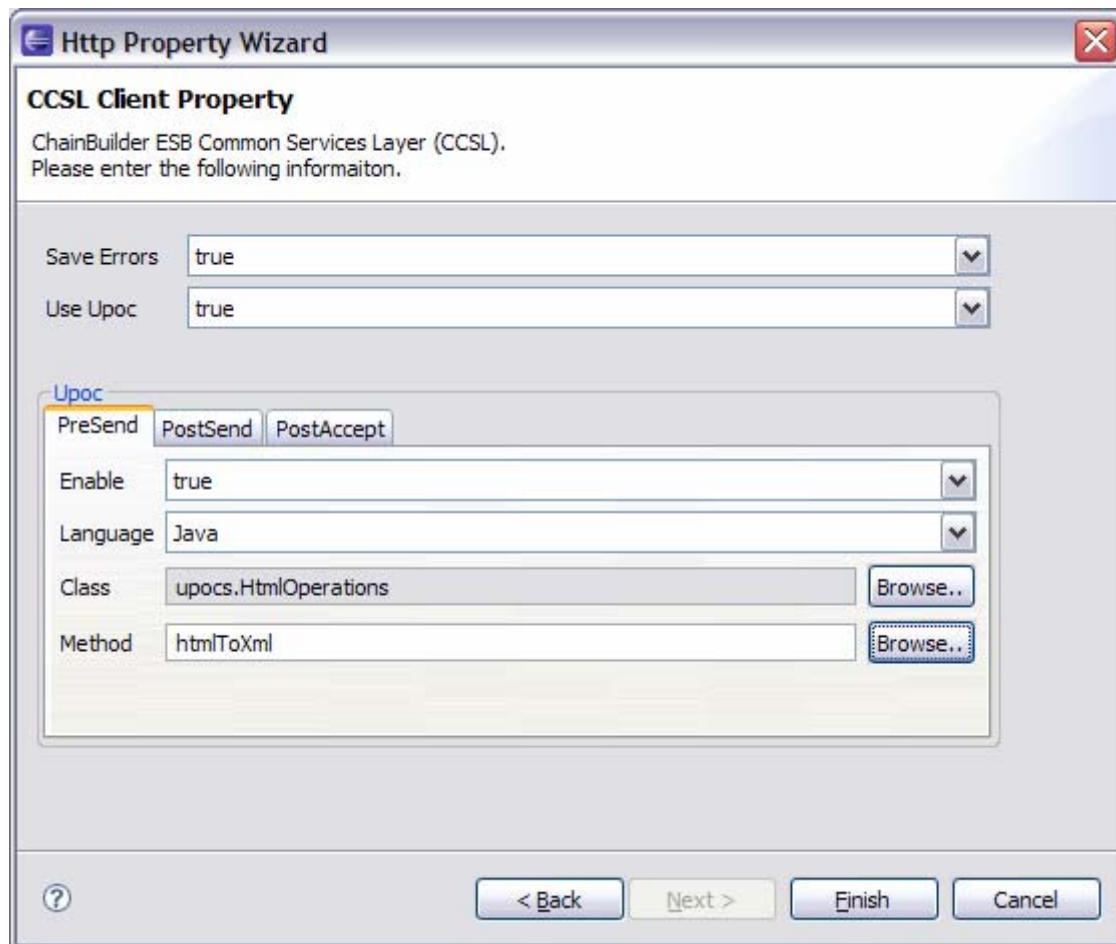
4.2. Using Custom Code

Before you can use a custom Java class, you must compile it. Right-click on the project name and select “ChainBuilder ESB→Build”



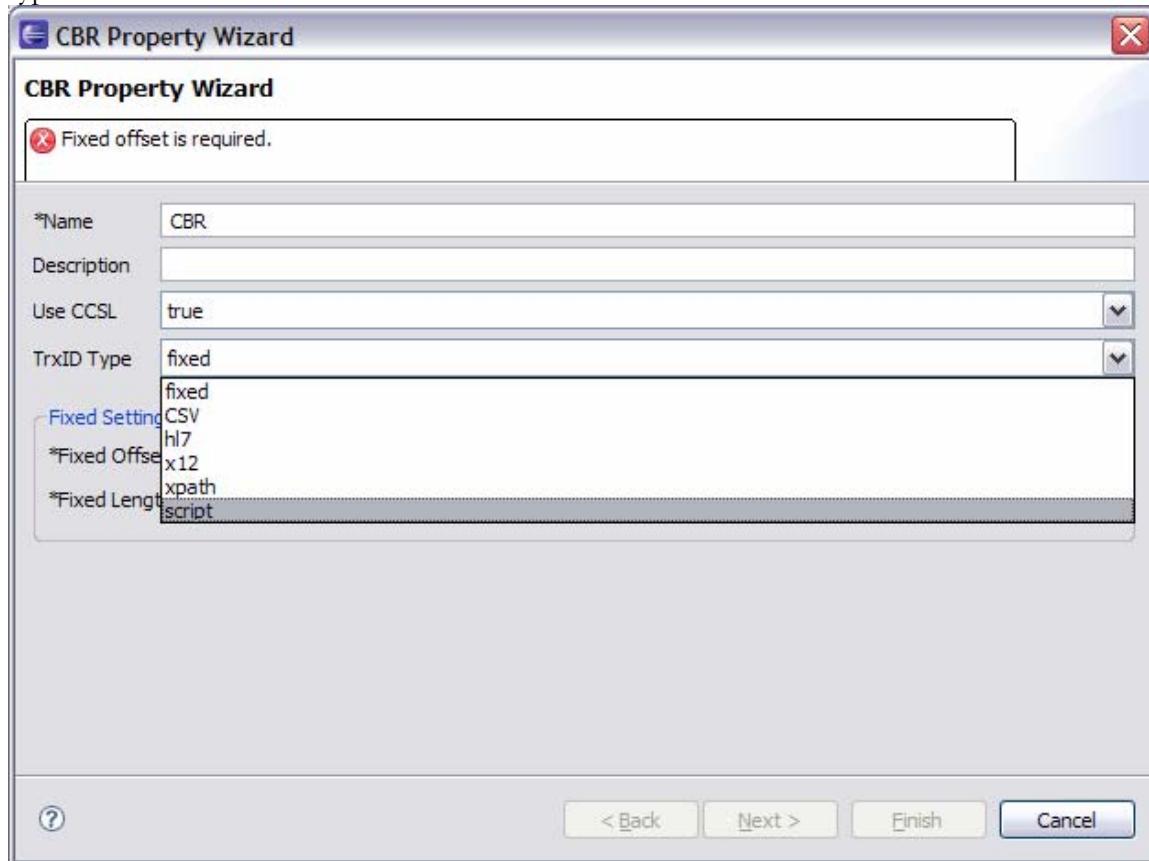
4.2.1. User Points of Control (UPOC)

User Points of Control are places where user code can be placed into the message flow between components. See the reference guide for a detailed description of UPOCs. Every component's wizard has a UPOC selection screen as shown here. Set “Use Upoc” to true. Then enable the appropriate contexts and select the language, class and method.

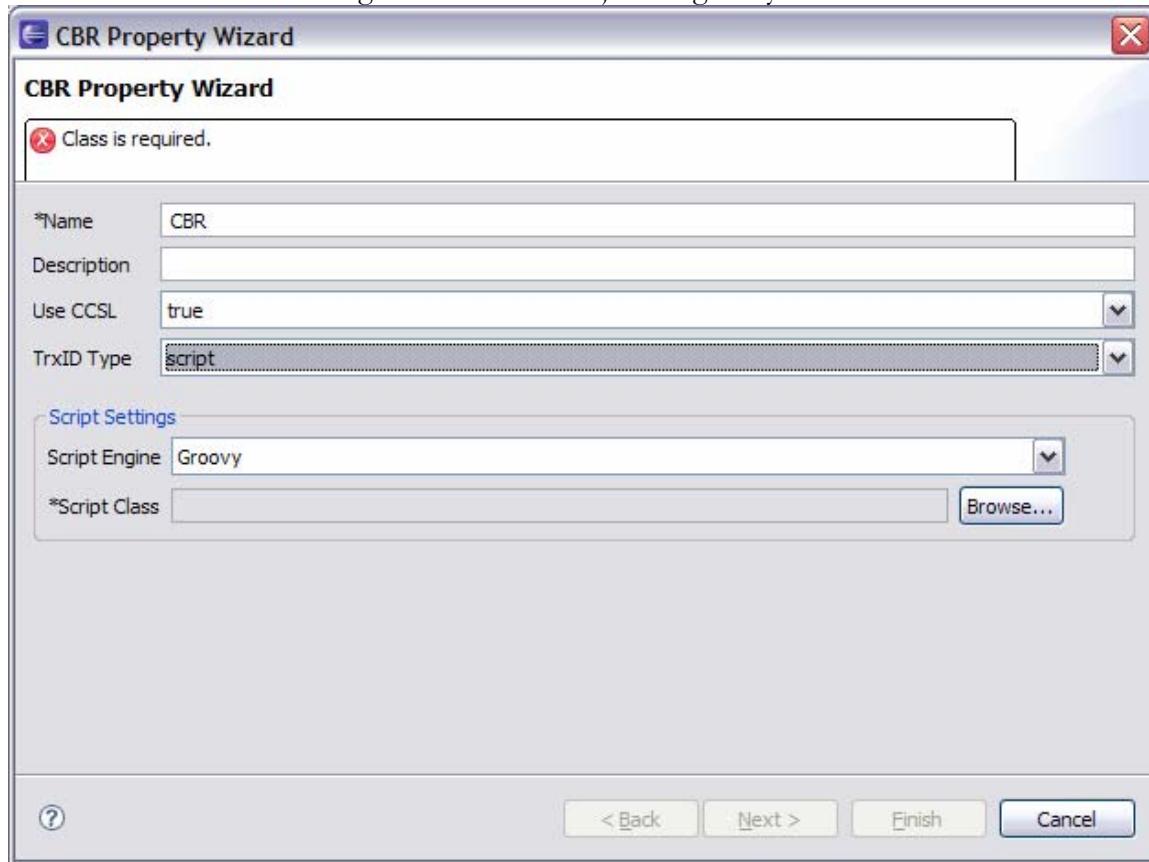


4.2.2. Transaction ID (TrxId)

The transaction ID is a string identifier used by the content based router to direct message exchanges. You can use custom code for the TrxId determination be select “script” as the type.

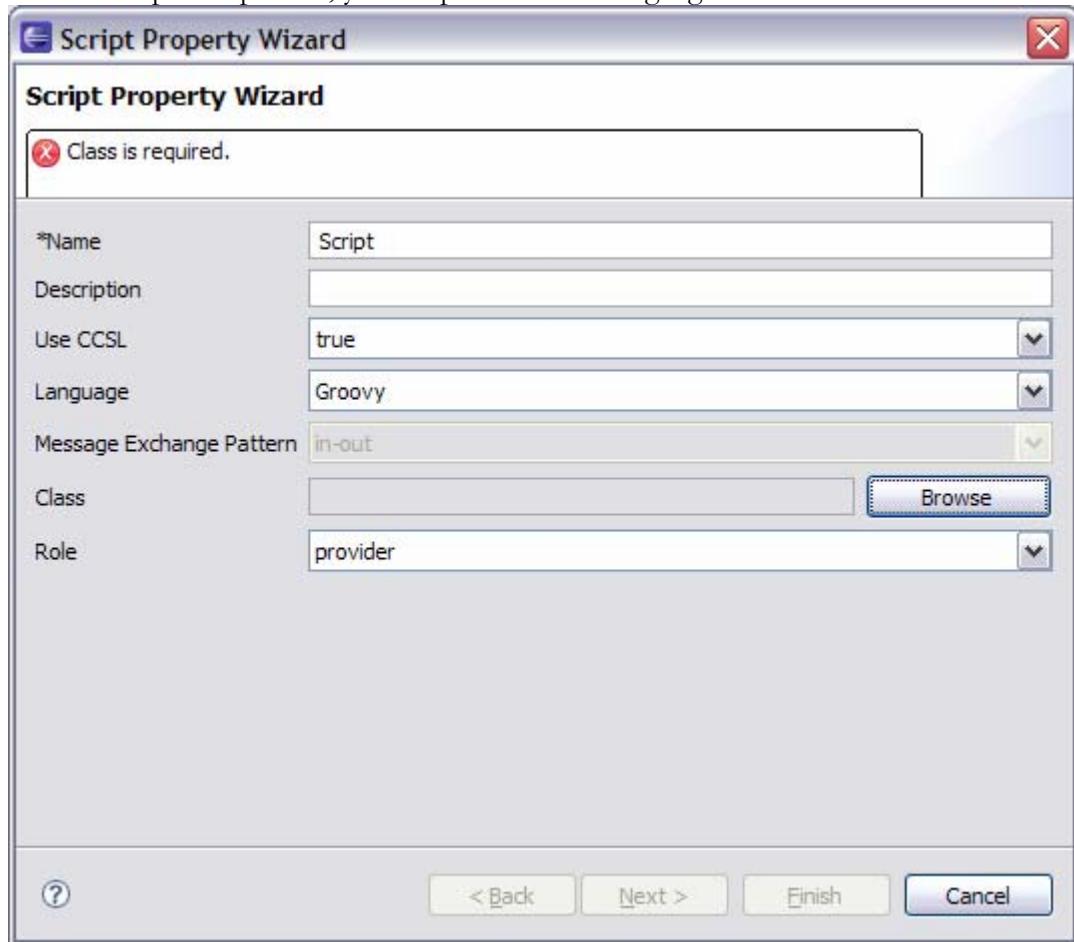


The you select the language and class. The TrxId determination class implements ITrxIdInterface which defines the methods that it must implement. The custom code wizard described in section 4.1 will generate a skeleton java or groovy file.



4.2.3. Script Component

For the script component, you simple select the language and class.



5. Detailed Component Descriptions

Each Binding Component and Service Engine has specific settings. After a Service Unit is created by adding a component to the canvas, its settings are displayed in the Properties Panel. All components have the following properties:

Name	The name of the Service Unit. This must be unique within the Service Assembly.
Description	A brief description of the Service Unit.
Interface Name	The name of the Interface that the Service Unit provides.
Service Name	The name of the Service provided by the Service Unit.

There is also an setting called “Use CCSL” to control whether the CCSL is used to change the standard behaviors a component. Please refer to the ReferenceGuide for additional information about ChainBuilder ESB Common Service Layer (CCSL).

The component-specific settings for each type of component are described below.

5.1. HTTP Binding Component

The settings for the HTTP Binding Component are as follows:

Mode	Available settings are <i>CLIENT</i> , <i>SERVER</i> or <i>BOTH</i> . This setting determines if the binding component will act as an HTTP Client (a JBI Provider end point), an HTTP Server (a JBI Consumer end point), or both.		
Client	End Point Name	The name of the Client end point	
	Default MEP	The only setting is <i>INOUT</i> .	
	Location URL	The HTTP URL of the target service.	
	SOAP	Enabled	Available values are <i>TRUE</i> or <i>FALSE</i> . If set to <i>TRUE</i> , the component will parse the soap requests and send the content into the NMR.
		Imported WSDL	List of imported WSDL files. If Enabled is true, then the Location URL is determined using the selected WSDL.
Server	End Point Name	The name of the Server end point	
	Default MEP	Available settings are <i>INONLY</i> , <i>INOUT</i> or <i>RELIABLEIN</i> .	
	Location URL	The HTTP URL where this proxy endpoint will be exposed. The URL is usually something like " <i>http://0.0.0.0:8192/jbi/Service</i> ". The 0.0.0.0 IP address binds the server socket to all networks that the host is in. If you use <i>localhost</i> , you will only be able to access the URL from the same computer.	
	Default Operation	The default operation name to set on the JBI exchange. If not set, it defaults to the QName of the root xml element.	
	SOAP	Enabled	Available values are <i>TRUE</i> or <i>FALSE</i> . If set to <i>TRUE</i> , the component will parse the soap requests and send the content into the NMR.

5.2. File Binding Component

The settings for the File Binding Component are as follows:

Mode	Available settings are <i>READ</i> , <i>WRITE</i> or <i>BOTH</i> . This setting determines if the binding component will act as a File Reader (a JBI Consumer end point), a File Writer (a JBI Provider end point) or both.		
Read	End Point Name	The name of the Reader end point.	
	Default MEP	Available settings are <i>INONLY</i> , <i>INOUT</i> or <i>RELIABLEIN</i> .	
	Source Dir	Specifies the local file system path that will be scanned for files to read.	
	Stage Dir	Specifies the local file system path that will be used to stage files	

		during reading. When the file reader matches a file in the source directory, it is moved to the stage directory where it is read.
	File Pattern	Specifies a pattern to match against file names in the source directory. The pattern uses glob style wildcards. The default is * which matches all files. Examples: <i>LAB???.dat</i> <i>*.xml</i>
	Scan Interval	Value in milliseconds that determines how often the source directory is scanned for new data files. Default value is 5000 (5 seconds).
	Read Style	Available settings are <i>RAW</i> or <i>NEWLINE</i> . <i>RAW</i> - The entire file is one record. <i>NEWLINE</i> - Each line in the file is one record.
	Record Type	Available settings are <i>STRING</i> , <i>XML</i> or <i>BINARY</i> . <i>STRING</i> - Each record is character data. <i>XML</i> - Each record is well formed XML. <i>BINARY</i> - Each record is binary data.
	Records Per Message	Integer value that determines the number of records from a file will be placed in an individual Normalized Message. 0 indicates that all records in the file will be placed in a single message. Any value > 0 will be the maximum number of records placed in a single message. The default value is 0.
	Char Set	The character set to use to read in character data.
	File Completion	Action Determines the action to take when all data has been read from a file. Available settings are <i>DELETE</i> or <i>ARCHIVE</i> . <i>DELETE</i> - The file is deleted when finished reading. <i>ARCHIVE</i> - The file is moved to an archive directory when finished reading.
	Archive Directory	The local file system path where files should be archived.
	Archive File Pattern	Describes a file pattern to use to rename the file when being archived. This can be used to add a date/time stamp to the file. If the value is null, then the file is not renamed when it is moved to the archive Directory. The pattern may contain literal characters as well as the following macros that will be replaced with values at runtime: {i}DATE{i} - The system date formatted as yyyyymmdd {i}TIME{i} - The system time formatted as hhmmss {i}BASENAME{i} - The original file's base name (name without extension). {i}EXT{i} - The original file's extension. {i}COUNT{i} - An automatically incremented value that starts from 1 when the component is started.
	Hold	Enabled Available settings are <i>TRUE</i> or <i>FALSE</i> . If set to <i>TRUE</i> , and an error occurs while processing a file, the file will be moved to

			the Hold Directory
		Hold Directory	The local file system path where files should be archived.
Two Pass Mode	Enabled	Available settings are <i>TRUE</i> or <i>FALSE</i> . Two Pass Mode causes the component to check the size of the files in the Source directory, wait for a set interval and check the sizes again. Only files that did not change size during the interval will be processed. This is to prevent processing a file that is still being written to by an external application.	
	Interval	Value in milliseconds to wait between scans during Two Pass Mode.	
Reply	Reply Dir	The local file system path where files will be written if the specified MEP returns an OUT message.	
	Reply Charset	The character set to use to write character data.	
	Reply Write Style	Available settings are <i>RAW</i> or <i>NEWLINE</i> . <i>RAW</i> - Each record from a Normalized Message is written to an individual file. <i>NEWLINE</i> - Each record from a Normalized Message is written to the same file separated by a newline.	
	Reply File Pattern	Describes a file pattern to use to name the file when being written. This can be used to add a date/time stamp to the file. The pattern may contain literal characters as well as the following macros that will be replaced with values at runtime: {i}DATE{ - The system date formatted as yyyyymmdd {i}TIME{ - The system time formatted as hhmmss {i}BASENAME{ - The original file's base name (name without extension). {i}EXT{ - The original file's extension. {i}COUNT{ - An automatically incremented value that starts from 1 when the component is started.	
Write	End Point Name	The name of the writer end point.	
	Default MEP	Available settings are <i>INONLY</i> , <i>INOUT</i> or <i>RELIABLEIN</i> .	
	Dest Dir	The local file system path where completed files are placed.	
	Stage Dir	The local file system path where files are created and written to.	
	File Pattern	Describes a file pattern to use to name the file when being written. This can be used to add a date/time stamp to the file. The pattern may contain literal characters as well as the following macros that will be replaced with values at runtime: {i}DATE{ - The system date formatted as yyyyymmdd {i}TIME{ - The system time formatted as hhmmss	

		{BASENAME} - The original file's base name (name without extension). {EXT} - The original file's extension. {COUNT} - An automatically incremented value that starts from 1 when the component is started.
	Write Style	Available settings are <i>RAW</i> or <i>NEWLINE</i> . <i>RAW</i> - Each record from a Normalized Message is written to an individual file. <i>NEWLINE</i> - Each record from a Normalized Message is written to the same file separated by a newline.
	Char Set	The character set to use to write character data.

5.3. FTP Binding Component

The settings for the FTP Binding Component in Base Mode are as follows:

Mode	Available settings are <i>READ</i> , <i>WRITE</i> or <i>BOTH</i> . This setting determines if the binding component will act as an FTP Reader (a JBI Consumer end point), an FTP Writer (a JBI Provider end point) or both.	
Read	End Point Name	The name of the reader end point.
	Default MEP	Available settings are <i>INONLY</i> , <i>INOUT</i> or <i>RELIABLEIN</i> .
	Host	Specifies the host name or IP address of FTP server
	User	Specifies the user name to login to FTP server
	Password	Specifies the password to login to FTP server
	Command File	Specifies the XML based command file described in the Script Mode. See the Reference Guide for details.
	Connection Mode	Available settings are ACTIVE or PASSIVE
	Transformer Mode	Available settings are ASCII or BINARY
	Source Dir	Specifies the path on the FTP server that will be scanned for files to read
	Transfer Dir	Specifies the local file system path where files are downloaded to
	Stage Dir	Specifies the local file system path that will be used to stage files during reading. When the file reader matches a file in the source directory, it is moved to the stage directory where it is read.
	File Pattern	Specifies a pattern to match against file names in the source directory. The pattern uses glob style wildcards. The default is * which matches all files. Examples: <i>LAB???.dat</i> <i>*.xml</i>

	Scan Interval	Value in milliseconds that determines how often the source directory is scanned for new data files. Default value is 5000 (5 seconds).
	Read Style	Available settings are <i>RAW</i> or <i>NEWLINE</i> . <i>RAW</i> - The entire file is one record. <i>NEWLINE</i> - Each line in the file is one record.
	Record Type	Available settings are <i>STRING</i> , <i>XML</i> or <i>BINARY</i> . <i>STRING</i> - Each record is character data. <i>XML</i> - Each record is well formed XML. <i>BINARY</i> - Each record is binary data.
	Records Per Message	Integer value that determines the number of records from a file will be placed in an individual Normalized Message. 0 indicates that all records in the file will be placed in a single message. Any value > 0 will be the maximum number of records placed in a single message. The default value is 0.
	Char Set	The character set to use to read in character data.
	File Completion	Action Determines the action to take when all data has been read from a file. Available settings are <i>DELETE</i> or <i>ARCHIVE</i> . <i>DELETE</i> - The file is deleted when finished reading. <i>ARCHIVE</i> - The file is moved to an archive directory when finished reading.
	Archive Directory	The local file system path where files should be archived.
	Archive File Pattern	Describes a file pattern to use to rename the file when being archived. This can be used to add a date/time stamp to the file. If the value is null, then the file is not renamed when it is moved to the archive Directory. The pattern may contain literal characters as well as the following macros that will be replaced with values at runtime: {i}DATE{i} - The system date formatted as yyyyymmdd {i}TIME{i} - The system time formatted as hhmmss {i}BASENAME{i} - The original file's base name (name without extension). {i}EXT{i} - The original file's extension. {i}COUNT{i} - An automatically incremented value that starts from 1 when the component is started.
	Hold	Enabled Available settings are <i>TRUE</i> or <i>FALSE</i> . If set to <i>TRUE</i> , if an error occurs while processing a file, the file will be moved to the Hold directory
	Hold Directory	The local file system path where files should be archived.
	Two Pass Mode	Enabled Available settings are <i>TRUE</i> or <i>FALSE</i> . Two Pass Mode causes the component to check the size of the files in the Source directory, wait for a set interval and check the sizes again. Only files that did not

		change size during the interval will be processed. This is to prevent processing a file that is still being written to by an external application.
	Interval	Value in milliseconds to wait between scans during Two Pass Mode.
Reply	Reply Host	Specifies the host name or IP address of FTP server
	Reply User	Specifies the user name to login to FTP server
	Reply Password	Specifies the password to login to FTP server
	Reply Connection Mode	Available settings are ACTIVE or PASSIVE
	Reply Transformer Mode	Available settings are ASCII or BINARY
	Reply Dir	The path on the FTP server where files will be written if the specified MEP returns an OUT message.
	Reply Charset	The character set to use to write character data.
	Reply Write Style	Available settings are <i>RAW</i> or <i>NEWLINE</i> . <i>RAW</i> - Each record from a Normalized Message is written to an individual file. <i>NEWLINE</i> - Each record from a Normalized Message is written to the same file separated by a newline.
Write	Reply File Pattern	Describes a file pattern to use to name the file when being written. This can be used to add a date/time stamp to the file. The pattern may contain literal characters as well as the following macros that will be replaced with values at runtime: {DATE} - The system date formatted as yyyy-mm-dd {TIME} - The system time formatted as hh-mm-ss {BASENAME} - The original file's base name (name without extension). {EXT} - The original file's extension. {COUNT} - An automatically incremented value that starts from 1 when the component is started.
	End Point Name	The name of the writer end point.
	Default MEP	Available settings are <i>INONLY</i> , <i>INOUT</i> or <i>RELIABLEIN</i> .
	Host	Specifies the host name or IP address of FTP server

	User	Specifies the user name to login to FTP server
	Password	Specifies the password to login to FTP server
	Command File	Specifies the XML based command file described in the Script Mode. See the Reference Guide for details.
	Connection Mode	Available settings are ACTIVE or PASSIVE
	Transformer Mode	Available settings are ASCII or BINARY
	Dest Dir	The path on the FTP server where completed files are placed.
	Transfer Dir	Specifies the local file system path where files will be uploaded from
	Stage Dir	The local file system path where files are created and written to.
	File Pattern	Describes a file pattern to use to name the file when being written. This can be used to add a date/time stamp to the file. The pattern may contain literal characters as well as the following macros that will be replaced with values at runtime: {DATE} - The system date formatted as yyymmdd {TIME} - The system time formatted as hhmmss {BASENAME} - The original file's base name (name without extension). {EXT} - The original file's extension. {COUNT} - An automatically incremented value that starts from 1 when the component is started.
	Write Style	Available settings are <i>RAW</i> or <i>NEWLINE</i> . <i>RAW</i> - Each record from a Normalized Message is written to an individual file. <i>NEWLINE</i> - Each record from a Normalized Message is written to the same file separated by a newline.
	Char Set	The character set to use to write character data.

The settings for the File Binding Component in Script Mode are as follows:

Default MEP	Available settings are <i>INONLY</i> , <i>INOUT</i> or <i>RELIABLEIN</i> .
--------------------	--

5.4. JMS Binding Component

The settings for the JMS Binding Component are as follows:

Role	Available settings are <i>CONSUMER</i> , <i>PROVIDER</i> or <i>BOTH</i> . This setting determines if the binding component will act as a consumer, a provider or both. When configured as a consumer endpoint, it will read messages from a destination and create Normalized Messages and route to the NMR. When the Default MEP is set to <i>INOUT</i> , a reply message with its Correlation ID equal to the request message ID is put into the reply destination.
-------------	--

	When configured as a provider endpoint, it receives a Normalized Message from the NMR and creates a JMS message and puts it into the destination. In this case, if the Default MEP is set to <i>INOUT</i> , it will retrieve a reply message from the optional reply destination with matching correlation ID equal to request message ID.	
Consumer	End Point Name	The name of the consumer endpoint.
	Default MEP	Available settings are <i>INONLY</i> , <i>INOUT</i> or <i>RELIABLEIN</i> .
	JNDI Initial Context Factory	Default JNDI InitialContext factory. The default value is: <i>com.sun.jndi.fscontext.RefFSContextFactory</i>
	JNDI Provider URL	Default JNDI provider url. The default value is: <i>file:/C:/CBESB/jndiDir</i>
	Destination Style	Available settings are <i>QUEUE</i> or <i>TOPIC</i> . <i>TOPIC</i> is used to support pub/sub.
	Target Destination Name	The destination to retrieve messages from.
	Read Style	Available settings are <i>RAW</i> or <i>NEWLINE</i> . <i>RAW</i> - The entire JMS message contents is one record. <i>NEWLINE</i> - Each line in the message is one record.
	Record Type	Available settings are <i>XML</i> , <i>STRING</i> or <i>BINARY</i> . <i>XML</i> - Each record is well formed XML. <i>STRING</i> - Each record is character data. <i>BINARY</i> - Each record is binary data.
	Records Per Message	Integer value that determines the number of records from a JMS message that will be placed in an individual Normalized Message. 0 indicates that all records in the JMS message will be placed in a single normalized message. Any value > 0 will be the maximum number of records placed in a single normalized message.
	Char Set	Value is the name of the char set to use to read and write character data.
	Reply Destination Name	Only used when the value of Default MEP is <i>INOUT</i> . It specifies the destination to put reply messages with a correlation ID that matches the original request message.
	Write Style	Available settings are <i>RAW</i> or <i>NEWLINE</i> . <i>RAW</i> - Each record from a Normalized Message is put on the reply destination as a separate message. <i>NEWLINE</i> - Each record from a Normalized Message is put on the reply destination in a single message separated by a newline.
	JNDI Connection Factory Name	Default JNDI name to lookup the JMS Connection Factory
	Reply Timeout	Time in milliseconds to wait for a reply before failing. Any value less than or equal to zero means infinite wait.
	Service Name	The name of the consumer service.
Provider	End Point Name	The name of the provider end point.
	Default MEP	Available settings are <i>INONLY</i> , <i>INOUT</i> or <i>RELIABLEIN</i> .
	JNDI Initial Context Factory	Default JNDI InitialContext factory. The default value is: <i>com.sun.jndi.fscontext.RefFSContextFactory</i>
	JNDI Provider	Default JNDI provider url. The default value is: <i>file:/C:/CBESB/jndiDir</i>

	URL	
	Destination Style	Available settings are <i>QUEUE</i> or <i>TOPIC</i> . <i>TOPIC</i> is used to support pub/sub.
	Target Destination Name	The destination to put messages.
	Read Style	Available settings are <i>RAW</i> or <i>NEWLINE</i> . <i>RAW</i> - The entire JMS message contents is one record. <i>NEWLINE</i> - Each line in the message is one record.
	Record Type	Available settings are <i>XML</i> , <i>STRING</i> or <i>BINARY</i> . <i>XML</i> - Each record is well formed XML. <i>STRING</i> - Each record is character data. <i>BINARY</i> - Each record is binary data.
	Records Per Message	Integer value that determines the number of records from a JMS message that will be placed in an individual Normalized Message. 0 indicates that all records in the JMS message will be placed in a single normalized message. Any value > 0 will be the maximum number of records placed in a single normalized message.
	Char Set	Value is the name of the char set to use to read and write character data.
	Reply Destination Name	Only used when the value of Default MEP is <i>INOUT</i> . It specifies the destination to read reply messages with a correlation ID that matches the original request message.
	Write Style	Available settings are <i>RAW</i> or <i>NEWLINE</i> . <i>RAW</i> - Each record from a Normalized Message is put on the reply destination as a separate message. <i>NEWLINE</i> - Each record from a Normalized Message is put on the reply destination in a single message separated by a newline.
	JNDI Connection Factory Name	Default JNDI name to lookup the JMS Connection Factory
	Reply Timeout	Time in milliseconds to wait for a reply before failing. Any value less than or equal to zero means infinite wait.
	Service Name	The name of the provider service.

5.5. Transformer Service Engine

The settings for the Transformer Service Engine are as follows:

TRN File	Specifies the Map file to use for transformations. The TRN file is created by the ChainBuilder ESB Map Editor and is located in the ESB or JBI Service Assembly project's src/xlate directory.
Default MEP	The only setting is <i>INOUT</i> .

5.6. Parser Service Engine

The settings for the Parser Service Engine are as follows:

Parser Type	Available settings are <i>MDL</i> or <i>X12</i> . <i>MDL</i> - The message definition that will be used for parsing is a ChainBuilder ESB MDL file. MDL files are created using the Message Format Editor. <i>X12</i> - The message definition that will be used for parsing is an X12 definition. The X12 message could be one of the included standard messages, or a variant definition created using the X12 Variant Editor.
Message Definition	The location of the message definition to use to parse messages. For MDL parsing, the value should be the path to an MDL file. Currently, the first Message definition in the MDL file is used as there is no way to specify an alternate message definition. For X12 parsing, the value should be the path to an X12 message definition.
Default MEP	The available setting is <i>INOUT</i> .

5.7. XSLT Service Engine

The settings for the XSLT Service Engine are as follows:

XSLT Style Sheet	Specifies the path and name of the XSLT style sheet to apply to XML messages.
Default Mep	The available setting is <i>INOUT</i> .

5.8. Sequencer Service Engine

There is no addition setting for Sequencer Service Engine.

5.9. Content-Based Router (CBR) Service Engine

The settings for the CBR Service Engine are as follows:

TrxID Type	Fixed	Offset	The “Fixed” TrxId Type specifies to use the length and offset to determine TrxId for message exchange. The offset attribute specifies the starting index of message content to be extracted for TrxId.
		Length	Specifies length of the TrxID. For example, If the message exchange's attachment contain data “ORD00123”, and you specify the length as 3 and offset as 0, then the string “ORD” will be returned as TrxId.
	CSV	Delimiter	The “CSV” TrxId Type specifies to use the delimiter and index to determine TrxId for message exchange. The delimiter attribute specifies the delimiter used in comma separated format.
		Index	Specifies the index to determine TrxID. For example, if the message exchange's attachment contain data “john,smith,male,25”, and you specify the delimiter as

		“,” and index as 3, then the string “male” will be returned as TrxId.
X12		Specifies to extract X12 transaction type as the TrxId for the message exchange. It is assumed that the message exchange's content is in X12 format. For example, if the message exchange has the X12 270 data, the “270” will be returned as TrxId type.
HL7		Specifies to extract HL7 transaction type as the TrxId for the message exchange. It is assumed that the message exchange's content is in HL7 format. For example, if the message exchange has the ADT A01 data, the “ADT_A01” will be returned as TrxId type. This feature will be supported in future release.
XPath	Expression	Specifies the optional XPath expression to extract the TrxId. If the expression is not specified, no TrxID will be extracted; the routing rule must be defined to use XPath as expression. See the Reference Guide for an example.
Script	Type	Specifies to use the script as a way to extract the TrxID of the message content in a message exchange. The supported settings are <i>GROOVY</i> and <i>POJO</i> .
	Class	Specifies the name of script class.
	Method	Specifies the method name in a script to be executed. The method must return a <i>String</i> type.

5.10. Script Service Engine

The settings for the Script Service Engine are as follows:

Mode	Available settings are Consumer or Provider.	
Consumer	End Point Name	The name of the consumer end point
	Default MEP	Available settings are <i>INONLY</i> , <i>INOUT</i> or <i>RELIABLEIN</i> .
	Type	Specifies the script type. The supported settings are <i>GROOVY</i> and <i>POJO</i> .
	Class	Specifies the name of script class. For Groovy, it is the Groovy file name.
	TriggerTime	The timer to trigger the calling of the scripting method. The value of 1000 stands for 1000 miliseconds. The trigger time allows setting up a simple time driven consumer by implementing a <i>time()</i> method in the user's script or class.
Provider	End Point Name	The name of the Server endpoint
	Default MEP	The only available setting is <i>INOUT</i> .
	Type	Specifies the script type. The supported settings are <i>GROOVY</i> and <i>POJO</i> .
	Class	Specifies the name of script class. For Groovy, it is the Groovy file name.

5.11. JDBC Service Engine

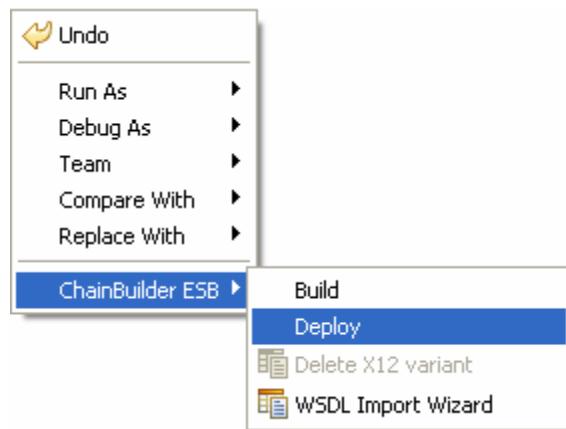
The settings for the Script Service Engine are as follows:

Driver	The fully qualified class name of the JDBC driver. For example: com.microsoft.jdbc.sqlserver.SQLServerDriver
URL	The driver specific URL that specifies the connection. For example: jdbc:Microsoft:sqlserver://SQLHost01:1433;databaseName=testdb
User	The user name to use to log into the database.
Password	The password to use to log into the database.
Request Handler	The handler class to use when processing request messages. It is responsible for parsing the request message to create an executable request object. A single instance of this class is used by each endpoint to process all requests. It defaults to "com.bostechcorp.cbesb.runtime.component.jdbc.processors.JdbcDefaultRequestHandler"
Exec Handler	The handler class to use when executing a request. Each session will have its own instance of this class so session specific data may be kept in the member variables. so session specific data may be kept in the member variables. It defaults to "com.bostechcorp.cbesb.runtime.component.jdbc.processors.JdbcDefaultExecuteHandler".
Auto Commit	If set to true, each successful request is committed automatically. If set to false, then the user is responsible for sending a Commit or Rollback to handle processing of transactions.
Connection Retries	When trying to establish a connection to the database, if there is a failure, it will make this many attempts to connect before erroring out.
Connection Interval	The number of milliseconds to sleep between reconnect attempts.
Transaction Timeout	The timeout in milliseconds to keep a transaction open before freeing the resources when there is no activity.
Default PageSize	The default number of rows to return in a single response. This may be overridden in the request message.

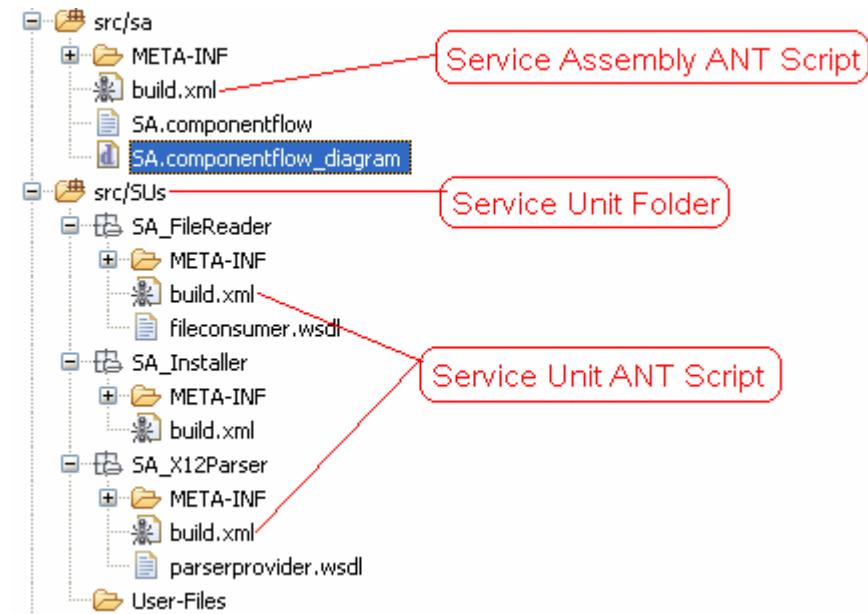
6. Generating the Service Assembly Archive

Once all of the Service Units are configured, the Service Assembly can be packaged for deployment. To generate the deployable archive, follow these steps:

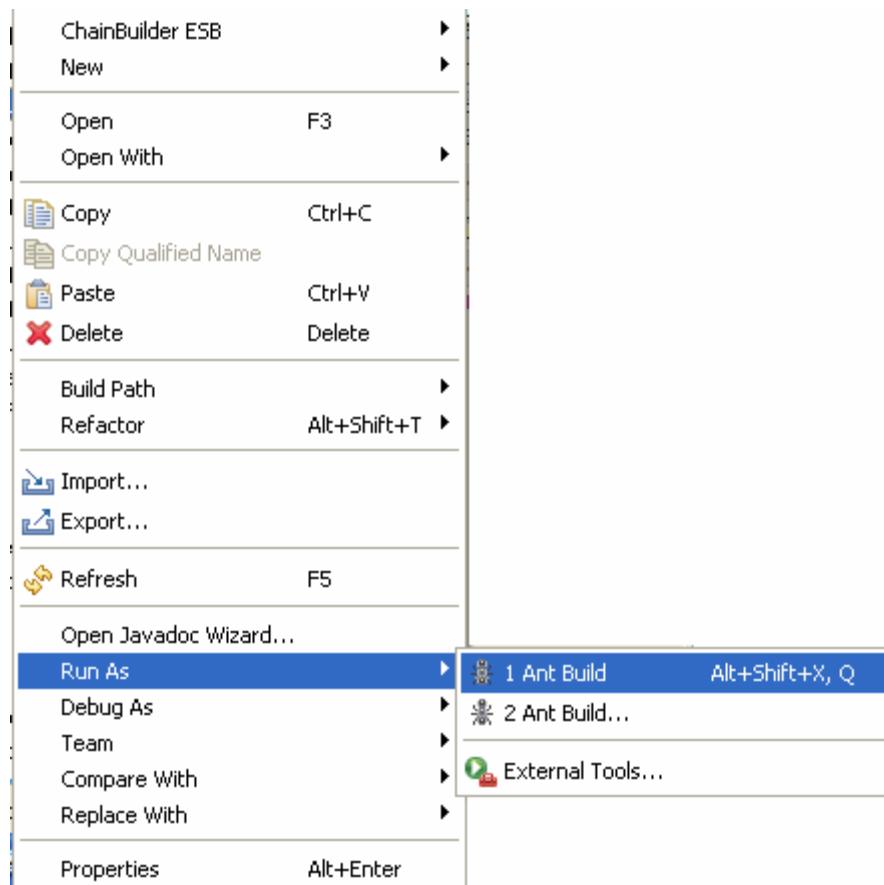
Right click on the Canvas area and select the deploy option.



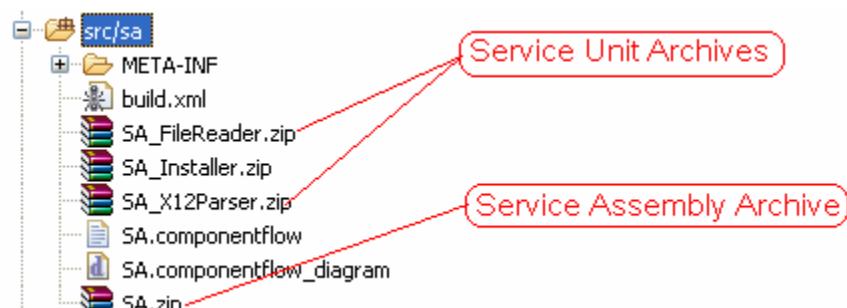
This will generate some new files and directories in the project. A new directory will be created in the `src` directory to contain the Service Units called `SUs`. Each Service Unit will have a directory containing the files it needs to be deployed at runtime.



Execute the Service Assembly ANT script by right clicking on it and selecting "Run As" and then selecting "Ant Build".



The Service Assembly ANT script will execute the individual Service Unit ANT scripts which will create a Service Assembly archive file for each Service Assembly and then create the Service Assembly archive file. These archives will be created in the project's `src/sa` directory. You may need to refresh the folder for them to be displayed. To refresh, select the `src/sa` folder and press the F5 key.



The Service Assembly archive has the same name as the JBI Service Assembly project. Each Service Unit archive has the same name as the Service Unit. The Service Assembly archive contains a copy of each Service Unit archive and is the only file needed for deployment.

Refer to the Reference Guide for instructions on how to deploy the Service Assembly archive.

7. ChainBuilder ESB Community

ChainForge.net is the internet's premier destination to share ChainBuilder and JBI knowledge with your peers.

Join the ChainBuilder ESB Community:

<http://www.chainforge.net/community>

As a member you can view content or contribute to a Forum:

<http://www.chainforge.net/community/forums.html>

Read ChainBuilder ESB related Blogs:

<http://www.chainforge.net/blogs>