

AnyLink Runtime Engine Server Guide

AnyLink 7



Copyright © 2019 TmaxSoft Co., Ltd. All Rights Reserved.

Document Information

Title: AnyLink Runtime Engine Server Guide

Publication Date: 2019-06-20

Software Version: AnyLink 7

Edition: v2.1.1

Website

<http://www.tmaxsoft.com>

Copyright Notice

Copyright © 2019 TmaxSoft Co., Ltd. All Rights Reserved.

Restricted Rights Legend

All TmaxSoft Software (Tmax AnyLink®) and documents are protected by copyright laws and international convention. TmaxSoft software and documents are made available under the terms of the TmaxSoft License Agreement and may only be used or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of TmaxSoft Co., Ltd.

Nothing in this software document and agreement constitutes a transfer of intellectual property rights regardless of whether or not such rights are registered) or any rights to TmaxSoft trademarks, logos, or any other brand features.

This document is for information purposes only. The company assumes no direct or indirect responsibilities for the contents of this document, and does not guarantee that the information contained in this document satisfies certain legal or commercial conditions.

The information contained in this document is subject to change without prior notice due to product upgrades or updates. The company assumes no liability for any errors in this document.

Trademarks

Tmax AnyLink® is registered trademark of TmaxSoft Co., Ltd. Other products, titles or services may be registered trademarks of their respective companies.

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses. : APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

Detailed Information related to the license can be found in the following directory :

`${AnyLink_HOME}\AnyLink-licenses`

Table of Contents

About This Document	vii
Chapter 1. Introduction	1
1.1. Overview	1
1.2. Key Functions	1
1.3. Server Configuration	4
Chapter 2. Runtime Engine Server	7
2.1. Server Components	7
2.1.1. Service Flow Engine	7
2.1.2. Adapter	8
2.1.3. Multi-binding Router	9
2.2. Server Architecture	9
2.2.1. System Architecture	10
2.2.2. Delivery Channel	11
2.2.3. Runtime Engine Service	13
2.3. Runtime Engine Error Processing	13
2.3.1. Delivery Channel	14
2.3.2. Service Flow	14
2.3.3. Adapter	15
Chapter 3. Runtime Engine Server Resource	17
3.1. Overview	17
3.2. Business Resources	17
3.2.1. Transactions and Transaction Groups	17
3.2.2. Environment Configurations for Transactions and Transaction Groups	21
3.2.3. Java Class Loader and Transaction Tree Class Loading Method	21
3.3. Libraries	23
3.4. Adapter Resources	23
3.5. Configuring Business System Information	24
3.5.1. Thread Pool	26
3.5.2. System Logging	27
3.5.3. Cluster	28
3.5.4. Deployment Policy	29
3.5.5. Encryption Algorithm	29
3.5.6. Debugging Mode	30
Chapter 4. Service Flow Engine	31
4.1. Service Flow Diagram	31
4.1.1. Basic Elements of Service Flow Diagrams	31
4.1.2. Service Flow Parallel Processing	32
4.1.3. Diagram Graphs and Block Structure	33
4.2. Basic Pattern	34

4.2.1.	Basic Flow Pattern	34
4.2.2.	More Split and Merge Patterns	36
4.3.	Service Implementation Method and Service Flow Type	42
4.3.1.	Proxy Service	42
4.3.2.	Composite Service	43
4.3.3.	Service Implementation	43
4.4.	Service Flow Variable and Activity Parameter	44
4.5.	Service Activity and Parameter	44
4.6.	Mapping	45
4.7.	Service Flow Expression	46
4.7.1.	Structured Expression	47
4.7.2.	Variable Expression	47
4.7.3.	Mapping Expression	48
4.8.	User Code and Handler	50
4.9.	Correlation	53
Chapter 5.	Multi-binding Router	55
5.1.	Overview	55
5.2.	Multi-binding Rule	55
5.2.1.	Value	56
5.2.2.	WeightBased	56
5.2.3.	TimeRange	57
5.2.4.	Handler	57
Appendix A.	Server APIs	59
A.1.	com.tmax.anylink.api Package	59
A.2.	com.tmax.anylink.api.serviceflow Package	60
A.2.1.	ActivityContext	60
A.2.2.	ActivityErrorHandler	61
A.2.3.	ActivityHandler	61
A.2.4.	BlockContext	62
A.2.5.	ErrorCodeMapper	63
A.2.6.	UserMapping	64
A.2.7.	ProcessContext	64
A.2.8.	ProcessHandler	65
A.2.9.	ServiceActivityHandler	66
A.2.10.	UserActivity	67
A.2.11.	VariableContext	67
A.3.	com.tmax.anylink.api.multibinding Package	68
A.4.	Default Abstract Class List	68
Index		71

List of Figures

[Figure 1.1]	Integration between AnyLink and Other Systems	2
[Figure 1.2]	External Integration	2
[Figure 1.3]	Role of AnyLink in an Enterprise System Architecture	3
[Figure 1.4]	Proxy Service Structure	3
[Figure 1.5]	Internal Linkage - Hub & Spoke Structure	4
[Figure 1.6]	AnyLink Development and Deployment Structure	5
[Figure 2.1]	AnyLink Components	7
[Figure 2.2]	AnyLink Service Flow Engine Triggering	8
[Figure 2.3]	Protocol Adapter	8
[Figure 2.4]	Runtime Engine Service Container Configuration	10
[Figure 2.5]	Runtime Engine I/O and Thread Architecture	11
[Figure 2.6]	Delivery Channel System Architecture	11
[Figure 2.7]	Request-Response-ACK Component Communication	12
[Figure 3.1]	Transaction Tree and Message Parsing	18
[Figure 3.2]	Symbolic Links and Java Class References	19
[Figure 3.3]	The Basic Hierarchy of a Java ClassLoader	22
[Figure 3.4]	ClassLoader Hierarchy Diagram Including an Example of the AnyLink Business ClassLoader	22
[Figure 3.5]	Error from a Symbolic Link Circular Reference	23
[Figure 4.1]	Events	32
[Figure 4.2]	Boundary Event	32
[Figure 4.3]	Gateways	32
[Figure 4.4]	Efficient Thread Processing Structure of Service Flow Engine without Wait Time	33
[Figure 4.5]	Elements of a Graph Structure in a Service Flow - Activities	33
[Figure 4.6]	Block Structure Elements of a Service Flow - Block Activity	34
[Figure 4.7]	Sequential Execution	35
[Figure 4.8]	Parallel Split and Synchronization	35
[Figure 4.9]	Exclusive Choice and Simple Merge	36
[Figure 4.10]	Multiple Choice and Synchronizing Merge	37
[Figure 4.11]	Join without Synchronization	38
[Figure 4.12]	Discriminator	39
[Figure 4.13]	Arbitrary Cycle	40
[Figure 4.14]	Implicitly Terminating Service Flows	40
[Figure 4.15]	Deferred Choice	42
[Figure 4.16]	Service Flow that Implements Proxy Services	43
[Figure 4.17]	Service Flow that Implements Composite Services	43
[Figure 4.18]	Service Flow in Service Implementation Type	43
[Figure 4.19]	Activity Input and Output	44
[Figure 4.20]	Service Activity Input	45
[Figure 4.21]	Mapping Editor	46

[Figure 4.22] Two Steps of Service Correlation	53
[Figure 5.1] Splitting a Service According to the Multi-binding Router Rule	55

About This Document

Intended Audience

This guide is intended for developers and system administrators who want to understand how the Tmax AnyLink[®] (hereafter AnyLink) system works.

AnyLink consists of a Data Integration Server (DIS) which manages deployment resources and environment configuration information, as well as a Runtime Engine (RTE) Server which implements services. This guide describes the internal architecture and components of the Runtime Engine Server. For more information about Data Integration Server, refer to the relevant guide.

Required Knowledge

Prior knowledge of the following is necessary to understand this guide.

- System and application integration.
- Servers (especially Java EE servers such as JEUS) that run on the Java Virtual Machine.
- Service oriented architecture.

Document Scope

This document does not cover detailed information about the core features of AnyLink such as integration tasks and methods. For more information, refer to documents related to EAI (Enterprise Application Integration) or ESB (Enterprise Service Bus) technologies.

This document does not describe Java EE or Java specs mentioned in this document. For more information, refer to relevant Java documents.

Document Organization

This guide consists of five chapters and one appendix as follows:

- Chapter 1: Introduction

Describes the basic functions and configurations of the AnyLink server.

- Chapter 2: Runtime Engine Server

Describes the AnyLink runtime engine server's components and architecture.

- Chapter 3: Runtime Engine Server Resources

Describes how to define and manage the resources required for executing the AnyLink runtime engine server.

- Chapter 4: Service Flow Engine

Describes the basic elements and patterns of diagrams created by using the AnyLink service flow engine.

- Chapter 5: Multi-binding Router

Describes how to split services in the multi-binding router and the router's rules.

- Appendix A : Server APIs

Describes the AnyLink user APIs.

Conventions

Convention	Meaning
<AaBbCc123>	File name of a program or source code
<Ctrl>+C	Hold the Control key and press C
[Button]	Button or Menu name
Boldface	Emphasis
" " (double quotes)	Reference to chapters or sections in the manual, or to other related documentation
'Input Item'	Description for an input item on the screen
Hyperlink	E-mail account, website, or a link to other chapters or sections
>	Progress order of menus
+----	Files or directories exist below
----	Files or directories do not exist below
Note	Reference or note
[Figure 1.1]	Figure name
[Table 1.1]	Table name
AaBbCc123	Command, execution result, or example code
[<i>command argument</i>]	Option parameter
< xyz >	Information between '< >' are changed to actual values
	Option (E.g. A B: One of A or B)
...	Repeated

System Requirements

Category	Requirement
Platform	Solaris 9-11
	HP-UX 11.x, 11i, 11iV2
	AIX 5L, 6L, 7L
	Linux Kernel 2.6 or later
	Windows 7 (32-bit, 64-bit)
Server	More than 1 GB RAM recommended (At least 512 MB)
	At least 500 MB hard disk space
Studio	Windows 7 (64-bit)
	1 GB RAM recommended (At least 512 MB)
	At least 500 MB hard disk space
Remote Agent	512 MB RAM recommended (At least 256 MB)
	At least 512MB hard disk space
Software	JDK 7.0
	JEUS 7 (Fix#3)
Supported Browsers for WebAdmin	IE 10 or later
	Chrome 41 or later
Supported Databases	Oracle 10g, 11g, 12c
	Tibero 6 FixSet03 or later

Chapter 1. Introduction

This chapter describes the basic functions and configurations of the AnyLink server.

1.1. Overview

AnyLink consists of two separate servers with different roles.

- Runtime Engine (RTE) server

Actually implements services and handles requests. It consists of the following components.

- Service Flow Engine

Executes service flows that define a flow of handling a request.

- Adapter

Supports connections to various applications through appropriate protocols. For more information about each adapter, refer to relevant guides.

- Multi-binding Router

Allows to dynamically perform routing between components at execution time

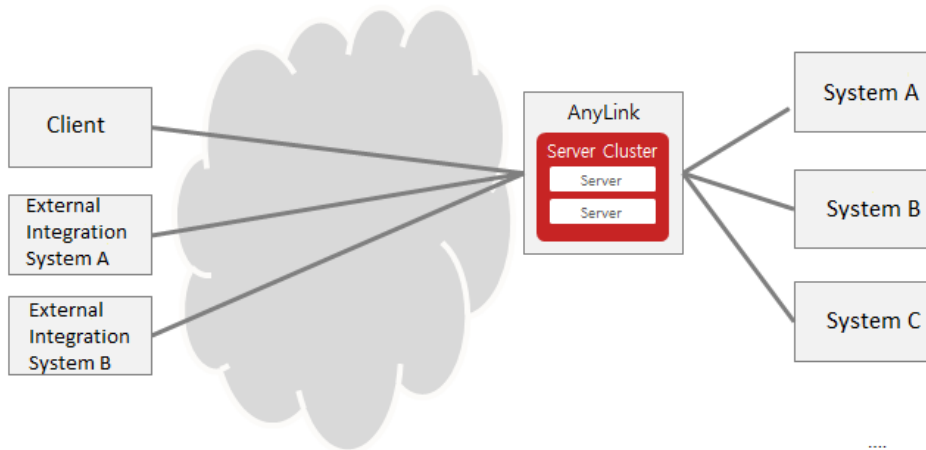
- Data Integration Server (DIS)

Manages resources to deploy and environment configuration information. It receives development resources from Studio, manages them, sends them to the runtime engine server, and manages the history. It also creates and compiles source code. It provides information for WebAdmin and Studio, and handles commands requested by WebAdmin and Studio. It requires an RDBMS to manage development resources and provide monitoring information such as log statistics.

1.2. Key Functions

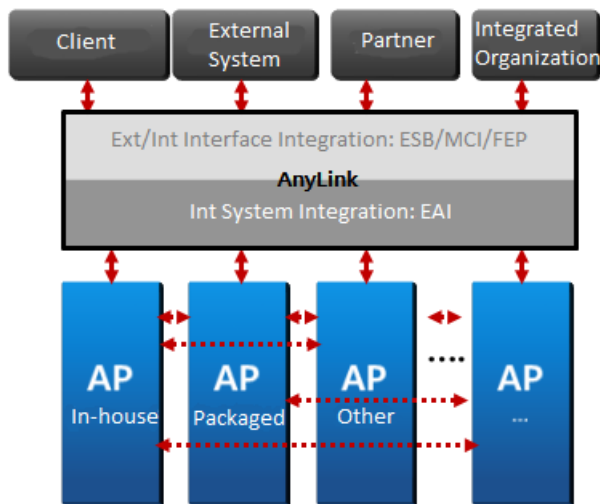
AnyLink is responsible for integration between different systems, and provides external services for various clients. AnyLink minimizes integration overhead in order to rapidly handle I/O tasks and minimize wait time. Also, it has protocols and application adapter functions for handling requests and responses of various types of servers and clients, as well as a function that orchestrates the flow of various services.

[Figure 1.1] Integration between AnyLink and Other Systems



AnyLink is responsible for internal integrations, as well as providing a layer of an integrated external interface which hides the details of an internal system.

[Figure 1.2] External Integration



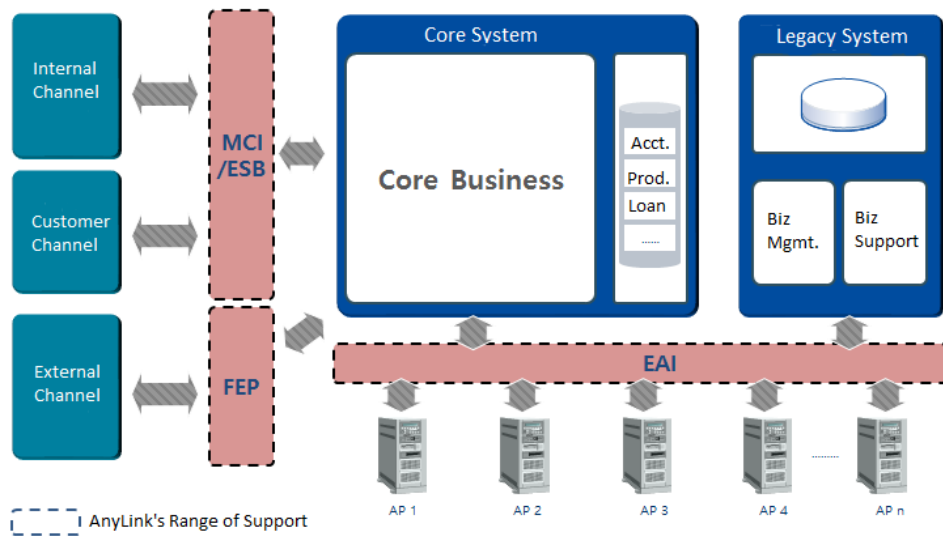
The following is a detailed description of the hierarchy role that AnyLink performs.

- Service Orchestration

AnyLink acts as a node that controls the entire enterprise system architecture.

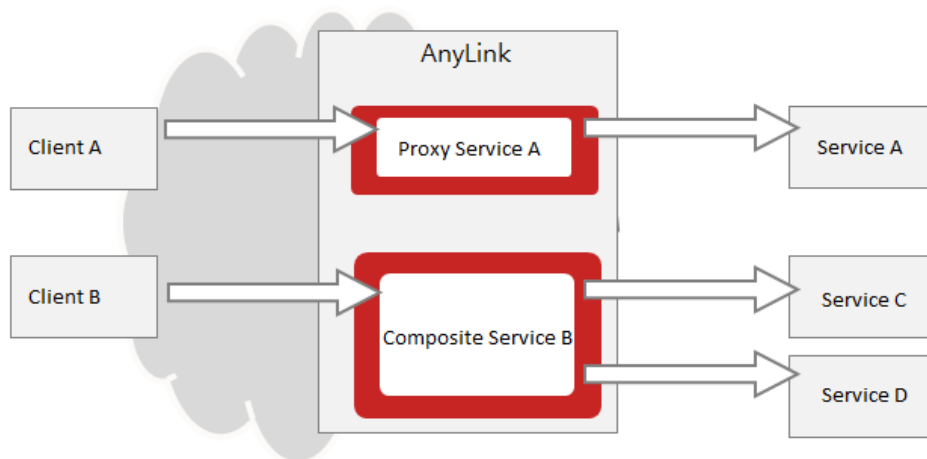
AnyLink is a control node that handles I/O operations and events, and is responsible for orchestration with each business process node.

[Figure 1.3] Role of AnyLink in an Enterprise System Architecture



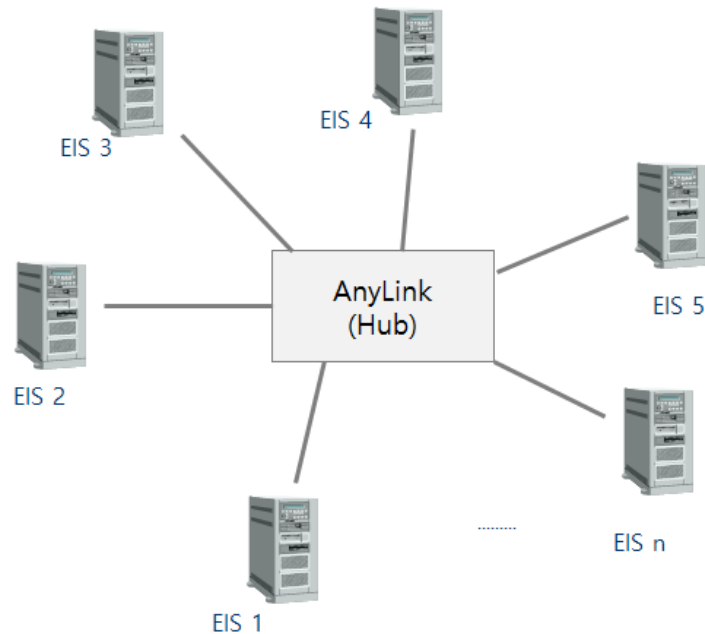
- Proxy Service
 - Wrapper Service
 - Composite Service

[Figure 1.4] Proxy Service Structure



- Linkage Service

[Figure 1.5] Internal Linkage - Hub & Spoke Structure

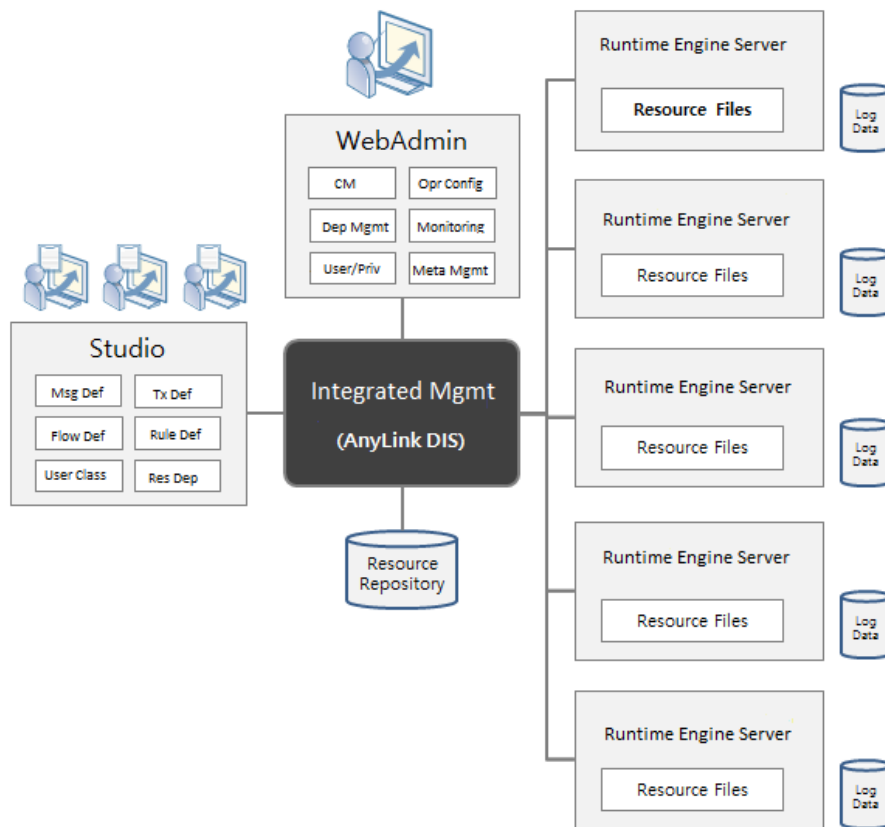


- Self Implemented Service
 - Integration logic and business logic are combined together

1.3. Server Configuration

AnyLink can be used in two types of environments: execution environments and development and deployment environments. An execution environment consists of the AnyLink runtime engine servers and target systems to be integrated, and has several structures depending on the configurations and structures of the target systems (refer to [\[Figure 1.4\]](#)). A development and deployment environment consists of the data integration server (for deployment), the runtime engine server (for execution), as well as WebAdmin (for operation management) and Studio (for rule creation and environment configuration).

[Figure 1.6] AnyLink Development and Deployment Structure



AnyLink separates the development and deployment server (Data Integration Server, DIS) and the operation server (Runtime Engine, RTE), so that in an actual operation environment, the RTE server can normally execute services regardless of whether the DIS is running or not.

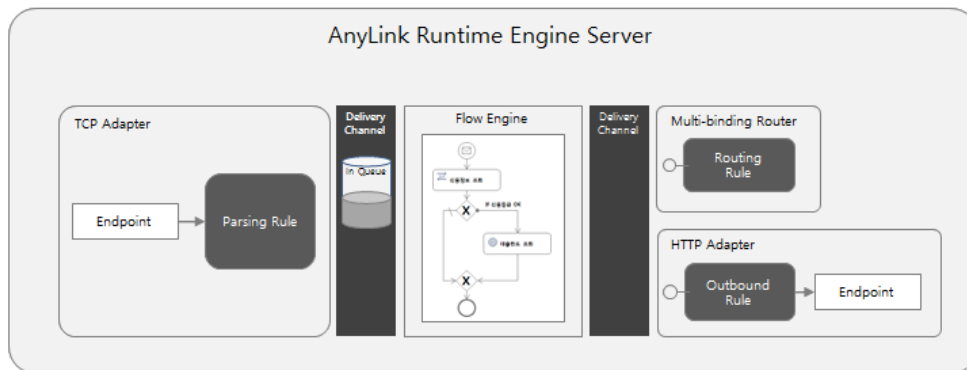
Chapter 2. Runtime Engine Server

This chapter describes the AnyLink runtime engine server's components and architecture.

2.1. Server Components

The AnyLink runtime engine server consists of a **service flow engine** (which determines the control flow), an **adapter** (which performs I/O operations for each protocol), and a **multi-binding router** (which handles multiple services as one).

[Figure 2.1] AnyLink Components



2.1.1. Service Flow Engine

The service flow engine executes a service flow when triggered by receiving an event that is a message sent by other components such as an adapter.

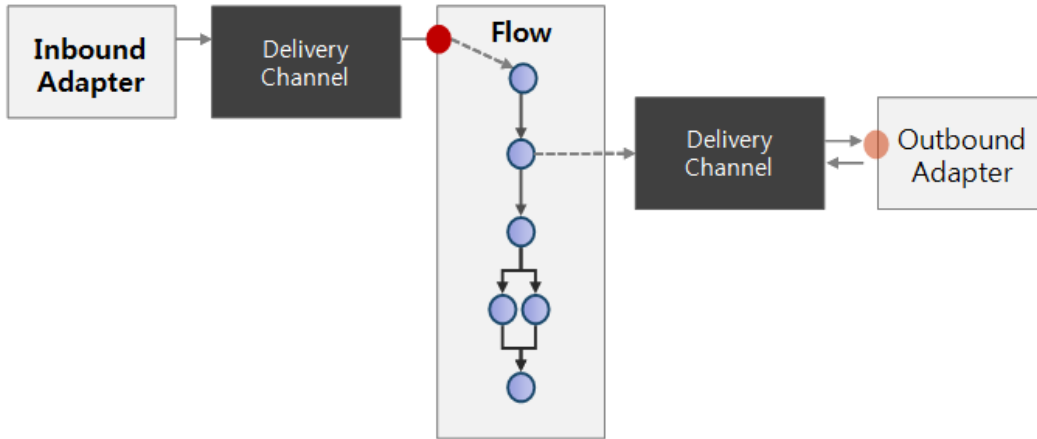
A service flow expresses a process flow by using a diagram borrowed from BPMN (Business Process Modeling Notation). BPMN is the modeling and notation standard that represents business processes defined in OMG (Object Management Group). It is good for expressing asynchronous and concurrent processes.

A service flow is defined in a diagram format (composed of activities and events) by using Studio, and additionally provides variables, expressions, mapping, handlers, and user activity functions, in addition to the process flow.

A service flow is created by a message sent by a component such as an adapter, and can call another adaptor or a service flow or send and receive a message. Also, it supports conditional splitting, selective execution, and concurrent execution, and has a variable for storing the status for controlling flows.

The AnyLink service flow engine has a special architecture that allows it to easily run asynchronous and concurrent processes, and executes them by using thread pools allocated in the engine.

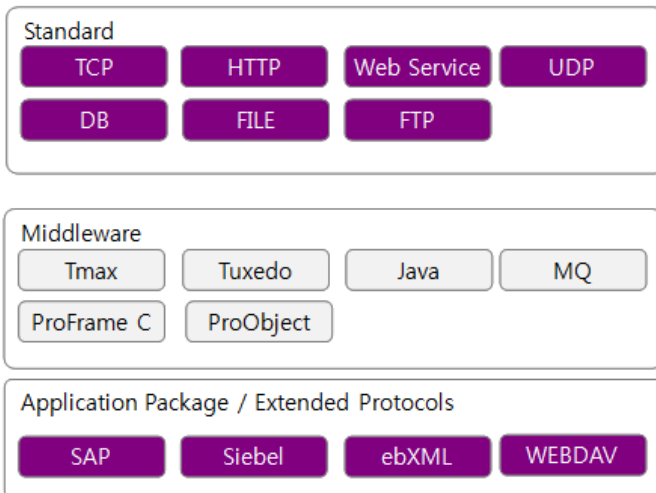
[Figure 2.2] AnyLink Service Flow Engine Triggering



2.1.2. Adapter

The adapter allows AnyLink to easily link with various applications, by using I/O processing through the protocol for a target application or by using a library of a target application.

[Figure 2.3] Protocol Adapter



The AnyLink Adapter can define two types of rules, inbound and outbound rules, depending on the direction of the I/O.

- The inbound rule sends requests from external systems to AnyLink, and returns responses to external systems.

- The outbound rule calls external systems from AnyLink, and receives responses.

Depending on the protocol, it may be difficult to distinguish between the request message of the inbound rule and the response message of the outbound rule. To efficiently handle this problem, AnyLink is designed to distinguish them by using a parsing rule that parses messages coming from an external system.

2.1.3. Multi-binding Router

The multi-binding router allows to dynamically perform routing between components at execution time. In AnyLink, a component triggers another component by sending a message.

AnyLink triggers other components by sending and receiving messages internally between components. Function names, request and response message types in a specific format, and error response message types are standardized in AnyLink as services.

AnyLink uses the following as internal services.

- Receive Message Events

Receive message events defined by service flows receive messages from the adapter.

- Outbound Rule

An adapter's outbound rule receives messages from service flows.

- Multi-binding Rule

Used to group different services together as one service. Message events of different service flows, different outbound rules of an adapter, or other multi-binding rules can be grouped together to be expressed as one service. A multi-binding rule can be used to split a group that consists of multiple services into specific services according to rules or conditions, or multi-cast to multiple services.

2.2. Server Architecture

The AnyLink runtime engine server consists of two large service components which are an adapter that communicates with external systems and a service flow engine in charge of processing logic, as well as a multi-binding router service component that provides routing between various services. It also has a **delivery channel** for inter-component communication and rule deployment, and a **resource manager**.

2.2.1. System Architecture

This section describes the runtime engine server's system architecture.

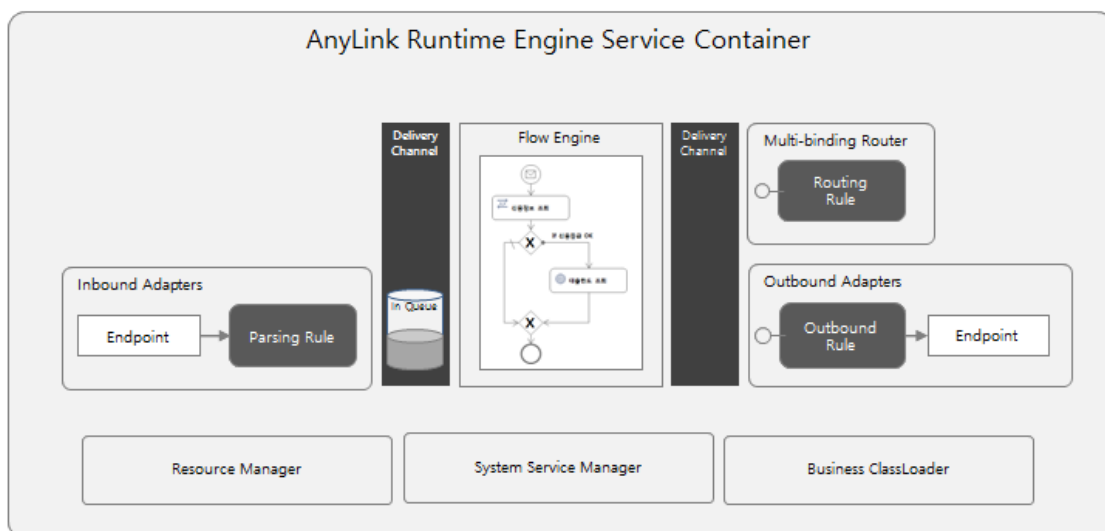
Runtime Engine Service Container

Runtime Engine Service Container (RTE Service Container) manages components' lifecycles as well as defines and provides services required for components. Also, it executes the service flow engine, multi-binding router, and adapter manager.

Main services provided are an internal communication channel called the delivery channel, the resource manager which manages deployment and loading of the service flow engine and adapter rules, the system resource manager which manages system resources such as thread pools and logs, and the Business ClassLoader which loads Java code used in various rules and flows.

Only one service flow engine component and multi-binding router component exist for each container, but there can be multiple adapter components for each adapter type. The adapter manager is responsible for starting and loading the adapter components. The multi-binding router component is located in the delivery channel and used to route services by using a method other than the service ID method.

[Figure 2.4] Runtime Engine Service Container Configuration



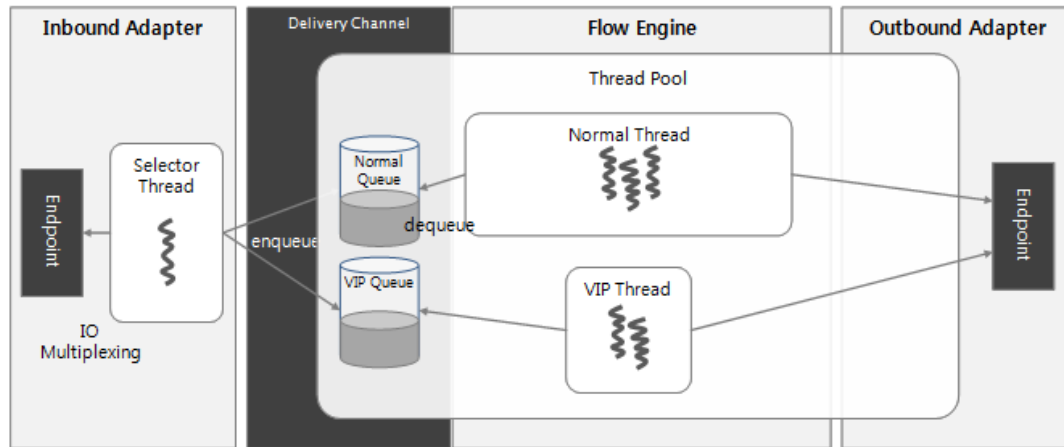
Engine Architecture

The AnyLink engine has an I/O structure which efficiently manages a large number of I/O channels, and an asynchronous process structure which minimizes waste from thread resources.

AnyLink's main communication adapter performs I/O processing in the I/O multiplexing method, and it implements the full non-blocking processing method so that other channels' processing is not affected by buffering of a specific I/O channel. The service flow engine, which uses thread pools, is implemented as

asynchronously as possible to minimize wait time. Therefore, even if multiple service flows are simultaneously executed, there is no lag from lack of thread pools.

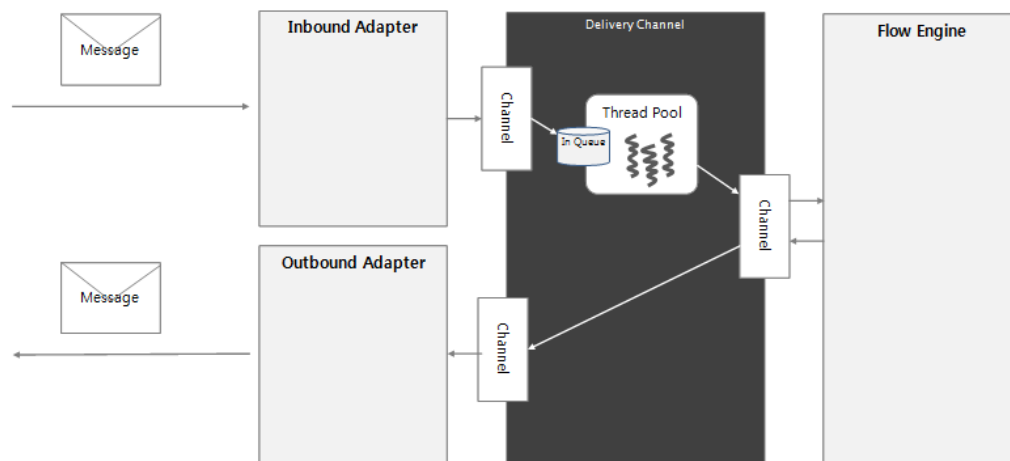
[Figure 2.5] Runtime Engine I/O and Thread Architecture



2.2.2. Delivery Channel

Delivery channel is a service that asynchronously supports internal communication between service components by using a pre-defined message pattern. It is the core architecture that reduces dependency between service components, and has the ability to use appropriate thread pools according to the messaging API.

[Figure 2.6] Delivery Channel System Architecture

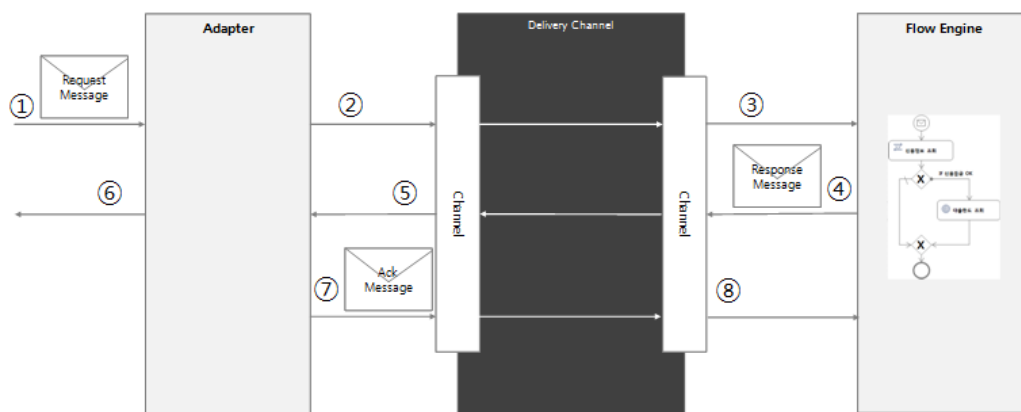


- **Messaging Pattern**

The delivery channel provides the following patterns of internal communication between service components.

Pattern	Description
Request-Response (2-way)	Sends a response message to a request message. The most common messaging pattern.
Oneway	Only sends a request message and does not wait for its response message.
Oneway-ACK	Sends an ACK message after a request message has been processed. Typically, the delivery channel guarantees the delivery and sends an ACK or a NAK message as a response.
Request-Response-ACK (3-way)	Sends an ACK or a NAK message to notify whether a response message to a request message has been properly handled.
Test-Message	Inquires whether a request message can be processed. Used for correlation matching routing in multi-binding routers. For information about correlation matching, refer to " 4.9. Correlation ".

[Figure 2.7] Request-Response-ACK Component Communication



- **Reliable Messaging and XA Messaging**

The delivery channel selectively supports reliable messaging and XA messaging when the OneWay messaging pattern is used.

Type	Description
Reliable Messaging	Stores messages in a permanent storage device such as a hard disk or RDBMS, and then sends an ACK message.
XA Messaging	Supports global transactions. Usually used to send multiple messages when global transactions are in progress in the Tmax adapter. Multiple messages grouped in a single transaction can be batch processed because global transaction messages such as prepare, commit, and rollback through 2-phase commit by using the Tmax adapter.

- **Transaction Propagation**

The delivery channel starts or ends transactions depending on the message properties, or propagates the transaction context. Also, it can manage global transactions' XA resources.

The delivery channel provides functions that can participate in transactions by receiving the transaction context of the message sent from a transaction manager such as Tmax or JEUS. As a transaction manager, it can also start transactions internally in AnyLink and propagate global transactions to XA resource managers such as external RDBMSs, or JEUS and Tmax. Each transaction's configuration can be set by using properties of a service flow or adapter rule.

2.2.3. Runtime Engine Service

The runtime engine service container provides services required for runtime execution in addition to the delivery channel.

- **Resource Manager**

Manages various rules, resources, and Java code contained in transactions and transaction groups of businesses that are internally defined.

- **System Resource Manager**

Manages the thread pools and system logging used by the runtime engine.

- **Business Class Loader**

Java code classloader that implements dependency processing such as the hierarchical structure of Java code and major resources, as well as the symbolic link.

- **Deploy Manager**

Dynamically and reliably deploys various resources received from the data integration server (DIS) through 2-phase deployment. If resources are undeployed or redeployed, then the existing resource files are moved to the backup directory, and either deleted or updated. If the runtime engine's deploy service generates an error in the first phase (prepare) of deployment, then the DIS rolls back the deployment to all servers.

2.3. Runtime Engine Error Processing

AnyLink provides a method for dealing with errors in various situations. Depending on the level in which an error is generated, the terms as well as error handling are different. This section briefly describes how to handle the errors that occur in the runtime engine.

2.3.1. Delivery Channel

There are instances where an error occurs when a message is sent from a service component to another service component from the delivery channel, which is a communication channel between service components inside the AnyLink runtime engine. This is an internal system error, so AnyLink users do not have to concern themselves of it, but it does help users to understand the internal operation of AnyLink.

The delivery channel that is responsible for communication between service components internally in the engine can distinguish between responses to requests and error responses. The delivery channel's error response is used to send an error as a response when an internal error occurs while handling a message in the service component.

The delivery channel's error response is different from abnormal responses from the business perspective. Abnormal responses from the business perspective must be sent via response messages, same as normal responses. Hence, when service components send abnormal response messages, they use the same APIs as the normal response messages in the delivery channel, and use a separate API to send an error.

2.3.2. Service Flow

The AnyLink service flow internally defines error events, error handlers, and error code mappers. Also, it defines mechanisms for handling abnormal responses for service requests.

- Error Events

The AnyLink service flow generates (when using an error event as the last event) or detects (when using an error event as a boundary event) error events for controlling flows.

Error codes are used for throwing or receiving error events. When an unexpected error (a Java exception) occurs during the execution of a service flow instance, the error code mapper converts the error to an error code. Hence, the error code mapper is an interface that converts a Java exception to an error code in which a service flow can recognize. By default, the error code mapper is not defined. However, for an exception that is not converted to an error code, the exception object's entire class name is used as the error code. An error code can be specified to handle an error by using a boundary event. If ALL is specified, all errors can be handled regardless of the exception type or error code value.

- Error Handlers

There is a process error handler which is specified for each service flow, and an activity error handler which is specified for each activity. When an exception occurs during the execution of a service flow instance, if a process or an activity error handler is specified, then the service flow can be normally executed because the error handler handles the error.

- Service Activity's Abnormal Responses

An abnormal response can be received when a service such as an adapter outbound rule or multi-binding router rule is invoked. An abnormal response is contextually defined on the side that sends the abnormal response. When an abnormal response occurs, mapping for the abnormal response can be separately specified.

Take caution in that if an abnormal response mapping or abnormal response parameter is not specified in the activity when a service sends an abnormal response, then instead of using a response mapping or response parameter, an abnormal response message is not stored in a parameter.

An abnormal response in an AnyLink service flow has nothing to do with an error event. Hence, the abnormal response is not related to the flow control method that handles errors in the service flow.

2.3.3. Adapter

Adapters define abnormal response messages in addition to response messages in the parsing rules. As well, an adapter can define system error messages when internal system errors occur in the adapter's endpoint.

Adapters define request messages and response messages as well as abnormal response messages in the parsing and outbound rules. As well, an adapter has rules designed for specifying responses for various error scenarios such as system errors and format errors. For more information about how each adapter handles errors, refer to the relevant adapter guide.

Chapter 3. Runtime Engine Server Resource

This chapter describes how to define and manage the resources needed for the execution of the AnyLink runtime engine server.

3.1. Overview

The AnyLink runtime engine server defines and manages the following resources needed for execution.

- Business resources related to businesses or transactions.
- Adapter resources related to adapters and adapter endpoints.
- System settings information related to thread pools and system logging.

3.2. Business Resources

Business resources are resources related to businesses or transactions, and consist of resources included in transactions and transaction groups defined in Studio. These are elements that configure the business concept, and are divided into transaction groups which consist of multiple layers and transactions which are end nodes. In the runtime engine server, business resources are located under each server's home directory by default.

```
${server.home}/repository/services
```

3.2.1. Transactions and Transaction Groups

The following business resources can exist in transactions and transaction groups. These resources are made up of a single compressed file (with the .iar extension) under a specific directory that indicates transactions in the runtime engine server.

- **Definition Files for Transactions and Transaction Groups (XML Document Files with the .biztx Extension)**

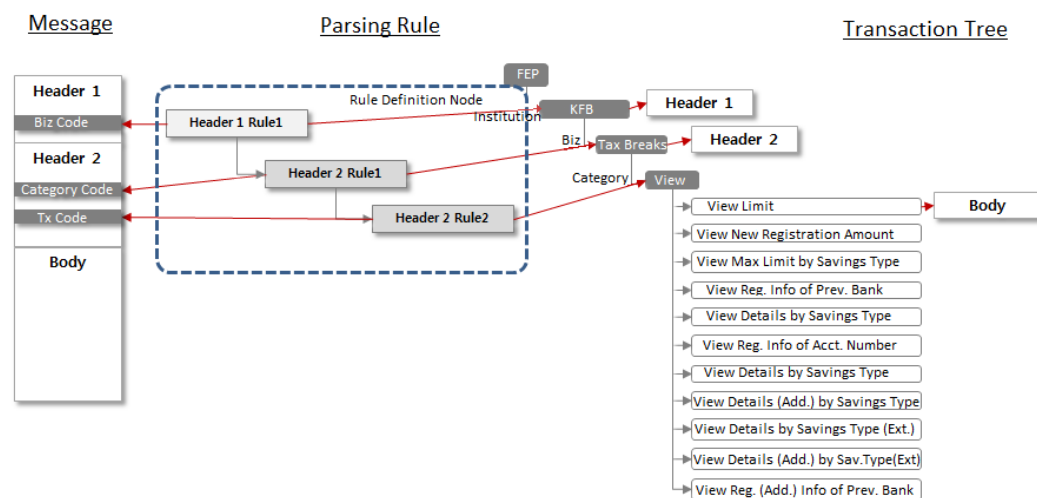
Transactions and transaction groups have a hierarchical tree structure in which a desired transaction can be found by parsing a request message. A transaction group is represented as a parent directory that contains sub transaction groups and transactions. Business resources related to the transaction

group are in the .iar archive file, just like a transaction. Transaction and transaction group information are in the XML document with the .biztx extension in the .iar archive file.

A transaction consists of a request message, a response message, and an abnormal response message, and inherits the information of transaction groups contained in the parent node.

A transaction group that is located in the parent node or the intermediate node of a transaction tree can have a sub-transaction identifier to find a sub transaction group or a transaction node (which is the end node) by parsing a request message.

[Figure 3.1] Transaction Tree and Message Parsing



The basic purpose of a transaction tree is to find a transaction that matches the information of a message by parsing a request message and to invoke a service (service flows, outbound rules, or multi-bound rules) of the AnyLink engine.

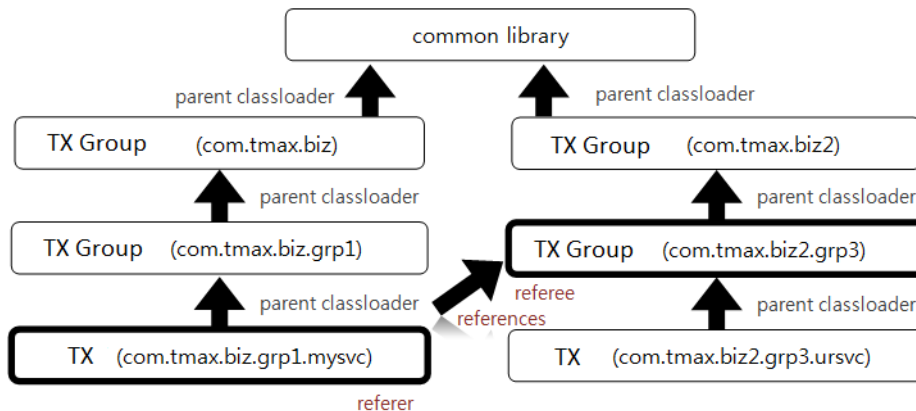
Transaction and transaction group information contain symbolic link information that is referenced by a transaction node or a transaction group node. A symbolic link links other transaction nodes that contain the resources referenced by the transaction node. Since basically only resources in a parent node can be referenced, a symbolic link is useful when sharing resources in transaction or transaction group nodes other than the parent node.

Among the resources that are managed together in AnyLink transactions or transaction group nodes, there are various resources that are converted to Java code. For example, messages, mappings, expressions, and various user classes are resources that can be used in Java code.

Java code is read through a class loader, and each AnyLink transaction tree is designed to have parent and child relationships in the class loader.

A symbolic link node is not a parent on a class loader, but the node can be used as if it were a parent thanks to the AnyLink ClassLoader implementation.

[Figure 3.2] Symbolic Links and Java Class References



- **Service Flow Definition File (XML Document Files with the .sfdl Extension)**

A service flow definition file is drawn by using a BPMN (Business Process Modelling Notation) diagram in AnyLink Studio's service flow editor.

Service flows are managed in the service flow engine, and the message events that are received are registered as AnyLink's internal services.

In the most basic form of AnyLink transactions, messages are typically received through an inbound adapter, and the messages are parsed through a transaction tree for identifying internal services. Typically, an internal service of AnyLink is a message event of a service flow, and it proceeds according to the flow defined in the service flow, and then calls the service activity of the service flow. Service activities drawn in a flow are mostly adapter outbound rule services.

The following describes the most basic form of an AnyLink transaction: an external system is called though an outbound rule service and a response is received. The response is then passed to the calling flow and the response message is sent to the adapter that sent the initial message.

- **Parsing Rule**

Parsing rules, which are rules for parsing external messages, are not managed as separate resources but instead are defined in transactions and transaction groups.

- **Adapter Outbound Rule (XML document files with a .orule extension)**

A document that defines outbound adapter rules used to send requests from AnyLink to other systems. The configuration information differs according to the protocol and implementation of each adapter. Basic configuration information consists of request messages, response messages, abnormal response messages, and adapter and endpoint IDs.

An outbound rule is one of AnyLink's internal services. If an outbound rule service is called internally from another component, it identifies the adapter and endpoint that are set in the rule, and sends messages to an external system.

- **Multi-binding Rule (XML Document Files with the .mbind Extension)**

If an internal service of AnyLink needs to be routed differently depending on the situation, multiple services can be grouped together as a single group. This type of service is a multi-binding router rule. The following can be configured: service splitting, round robin, weight-based or time-based round robin, service flow correlation, handlers, etc. Request, response, and abnormal response message mapping (message conversion) can also be specified.

- **Message (XML Document File with the .Umsg Extension)**

Multiple messages can exist inside a .Umsg file. AnyLink distinguishes two different types of messages: DTO (Data Transfer Object) which expresses the internal structure of the data used for input and output, and messages in specific formats. A .Umsg file can contain a single DTO and messages in many different formats. The messages must have the same structure with the DTO. The supported message formats include XML, fixed length, delimiter, NameValue, and JSON.

AnyLink converts DTOs and messages into Java code and manages it. This is for optimizing performance and efficiency. This means that the .class files generated by compiling .Java source files that correspond to .Umsg documents are included in transaction resources.

- **Message Mapping (XML Document Files with the .Map Extension)**

A message mapping document defines the rules for converting input messages to output messages. AnyLink specifies conversion rules in the form of connecting the fields of each input message and output message, and can use various expressions as mapping sources in addition to the input message.

To improve the execution speed of mapping that consists of a combination of input/output fields or expressions, AnyLink internally converts mapping to Java code and manages it. Hence, just like message files, the .class file generated by compiling .Java source file is included in the transaction resource.

- **Expression (XML Document Files with the .expr Extension)**

AnyLink uses expressions to indicate a part of a message or express a function expression.

An AnyLink expression can express the structure of a message. For example, if a field named field 1 exists in msg1, it is represented by the expression "msg1.field1". In a service flow, the user can access a DataField variable which is declared in a flow. For example, to access a field named "name" of a variable named dataField1, the user uses the expression "\$dataField1.name".

In an AnyLink expression, the user can use parentheses to represent groups, and call Java methods. For example, in the variable example above, if a field named "name" corresponds to the Java String class, the user can extract only the first two characters as follows.

```
"$dataField1.name.substring(0, 2)"
```

Optionally, the user can explicitly specify the Java type of the variable. For example, if a variable named `dataField1` has a datatype of `com.example.NamedPerson`, the user can rewrite the above expression as follows.

```
"$dataField1<com.example.NamedPerson>.name.substring(0,2) "
```

AnyLink converts expressions to Java code by considering the execution speed. Incorrect expressions will cause errors when converting and compiling Java code during deployment in Studio. Same as message and mapping files, `.class` files generated by compiling `.expr` files are included in transaction resources.

In AnyLink, expressions are used in splitting condition statements of service flows, correlation calculations, and multi-binding router rules.

Expressions that can be used to represent the source information of a message mapping have a slightly special form in order to support mapping characteristics, such as creating an empty object if there is an object reference of an empty source, and supporting array type mapping.

3.2.2. Environment Configurations for Transactions and Transaction Groups

The following describes the environment configuration files of transactions and transaction groups.

- **Environment Configuration for Transactions and Transaction Groups (XML document files with the `.bizcfg` extension)**

Environment configuration documents of transactions and transaction groups are not directly related to transaction information, but stores information related to the transaction environment.

Transactions and transaction group resources are managed in a single `.iar` file for each node, but the environment configuration document files of transaction nodes exist separately in a document format with the `.bizcfg` extension under the directory that indicates the transaction node or transaction group node.

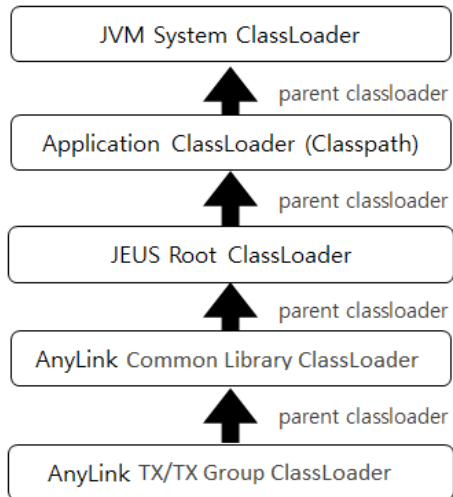
The environment configuration document can set the transaction and outbound rule trace log levels. The user can also specify thread pools to be used in service flows of corresponding transaction nodes.

3.2.3. Java Class Loader and Transaction Tree Class Loading Method

Business resources that indicate transactions and transaction groups contain automatically created Java code such as messages, mappings, and expressions, in addition to Java code such as user classes and handlers.

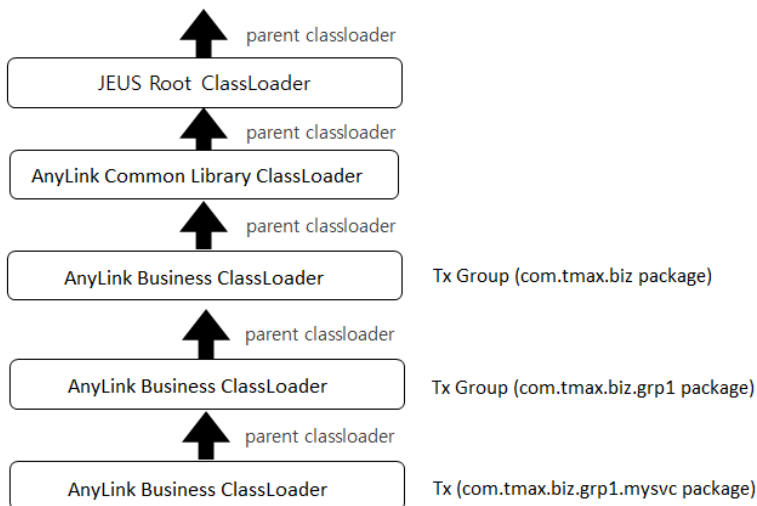
As mentioned earlier in explaining the symbolic link, each Java code follows the code reference structure by class loader configuration according to the characteristics of Java. AnyLink is executed in the Java EE environment, so a basic class loader has the following structure.

[Figure 3.3] The Basic Hierarchy of a Java ClassLoader



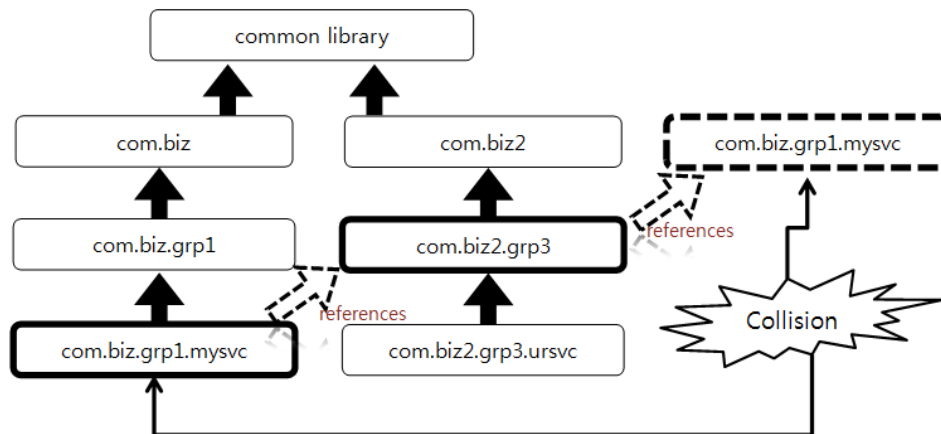
The AnyLink business ClassLoader is not a single ClassLoader, and has child ClassLoaders in a transaction tree format under the root ClassLoader.

[Figure 3.4] ClassLoader Hierarchy Diagram Including an Example of the AnyLink Business ClassLoader



Due to the nature of the ClassLoader hierarchy, circular references across multiple layers are not allowed. This is also true when using symbolic links. Symbolic links create a type of parent ClassLoader-like relationship.

[Figure 3.5] Error from a Symbolic Link Circular Reference



If a specific transaction or transaction group node's information is modified and redeployed, then the node and all its child nodes are reloaded. The child nodes here contain logical child nodes that are referenced using a symbolic link.

3.3. Libraries

If there is a common Java library in an AnyLink transaction tree, then this library is managed in the library directory, separate from the transaction tree.

In the AnyLink business classloader hierarchy, a library is loaded by the top business classloader. When a library is deployed at the time of operation, due to the nature of loading the library code to the top level node of the classloader, the classloader of the entire transaction tree including the child nodes is loaded again.

By default, the library files are located under each server home directory on the runtime engine server.

```
${server.home}/repository/lib
```

3.4. Adapter Resources

The AnyLink runtime engine manages resources such as adapter communication endpoints and configuration information needed for adapters responsible for communication with various protocols and applications. Major adapter-related resources are adapter configuration information, endpoints, and endpoint groups.

By default, adapter resources are located under each server home directory on the runtime engine server.

```
${server.home}/repository/adapters
```

The following are the types of adapter resources.

- **Adapter**

The adapter configuration information file is an XML document with the .adt extension, and sets information such as protocol, thread pool ID, trace log level, and clustering information.

Note

AnyLink can set and generate trace logs and transaction logs by task. These logs are handled by the log adapter. For more information, refer to the adapter guide.

- **Endpoints and Endpoint Groups**

An endpoint information file is an XML document with the .ep extension, and sets information such as communication direction, synchronization mode, transaction node ID, connection timeout, thread pool ID, trace log level, outbound endpoint log level, clustering information, and system error message.

An endpoint group groups together multiple endpoints, and is set in an XML document with the .epg extension. Set the method for splitting into child endpoints and endpoint groups. For example, routing methods such as round-robin, minimum-request, handler, and dedicated round-robin are supported.

An endpoint group is represented by a parent directory with a child endpoint group or endpoint.

3.5. Configuring Business System Information

A business system is configured with a single runtime engine server or a runtime engine server cluster.

Servers included in a cluster of Java EE servers (which are the basis of AnyLink) belong to the same business system. In AnyLink, transaction tree information is shared within a business system. Therefore, the basic deployment unit of an application executed in AnyLink can be thought of as a business system unit. Hence, businesses such as transactions that are deployed differ according to the business system.

AnyLink runtime engine's system settings are divided into common settings for business systems and settings for each individual runtime engine server. Mostly, thread pools, logging, Java EE server domain user ID, and password are set. If there are no special settings, then the default thread pool ID and service timeout are set. The user can set the basic trace log level of the system, as well as thread pool and queueing information in the file. The file also has various additional settings needed in the runtime engine server.

The section will describe the major configuration areas. For more information, refer to the description in the relevant section.

Business system configurations are saved in the following path for each server.

```
${server.home}/repository/bizsystem/bizsystem.config
```

<bizsystem.config>

```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:bizSystemInfo xmlns:ns2="http://www.tmaxsoft.com/schemas/AnyLink/reliableQueue/"

xmlns:ns1="http://www.tmaxsoft.com/schemas/AnyLink/common/"
xmlns:ns3="http://www.tmaxsoft.com/schemas/AnyLink/serverPlugin/"
xmlns:ns0="http://www.tmaxsoft.com/schemas/AnyLink/">
<ns0:sysId>IFL_SYS</ns0:sysId>
  <ns0:id>IFL_SYS</ns0:id>
  <ns0:name>IFL_SYS</ns0:name>
  <ns0:version>4</ns0:version>
  <ns0:registererDate>2014-10-13 12:54:03.514</ns0:registererDate>
  <ns0:registeringUser>admin</ns0:registeringUser>
  <ns0:modificationDate>2014-11-10 20:20:09.829</ns0:modificationDate>
  <ns0:user>administrator</ns0:user>
  <ns0:passwd>ba05778dea933f47bc150c743b122336</ns0:passwd>
  <ns0:bizSystemDefaultThreadPoolId>bizSystemDefaultThreadPool
</ns0:bizSystemDefaultThreadPoolId>
  <ns0:threadPool>
    <ns0:id>bizSystemDefaultThreadPool</ns0:id>
    <ns0:name>bizSystemDefaultThreadPool</ns0:name>
    <ns0:queueFullPolicy>SystemError</ns0:queueFullPolicy>
    <ns0:useVipThread>false</ns0:useVipThread>
    <ns0:threadPriority>-1</ns0:threadPriority>
    <ns0:vipThreadPriority>-1</ns0:vipThreadPriority>
    <ns0:commonSetting>
      <ns0:queueSize>-1</ns0:queueSize>
      <ns0:min>60</ns0:min>
      <ns0:max>100</ns0:max>
      <ns0:keepAliveTime>300</ns0:keepAliveTime>
      <ns0:vipThreadCount>10</ns0:vipThreadCount>
    </ns0:commonSetting>
  </ns0:threadPool>
  <ns0:systemLogging>
    <ns0:fileLocation>${server.home}/logs/AnyLink_rte_%d{1}.log</ns0:fileLocation>
    <ns0:logLevel>FINEST</ns0:logLevel>
    <ns0:logger>
      <ns0:name>com.tmax.anylink.dis.level</ns0:name>
      <ns0:logLevel>INFO</ns0:logLevel>
    </ns0:logger>
  </ns0:systemLogging>
  <ns0:nodeList>
    <ns0:node>
      <ns0:name>server1</ns0:name>
    </ns0:node>
  </ns0:nodeList>
  <ns0:clusterGroup>
```

```

<ns0:adapterClusterGroup>
  <ns0:adapterCluster>
    <ns0:clusterId>TCP_ADT</ns0:clusterId>
    <ns0:activeCount>2</ns0:activeCount>
    <ns0:construction>
      <ns0:serverId>server4</ns0:serverId>
      <ns0:priority>0</ns0:priority>
      <ns0:clusterMember>true</ns0:clusterMember>
    </ns0:construction>
    <ns0:construction>
      <ns0:serverId>server5</ns0:serverId>
      <ns0:priority>1</ns0:priority>
      <ns0:clusterMember>true</ns0:clusterMember>
    </ns0:construction>
    <ns0:construction>
      <ns0:serverId>server6</ns0:serverId>
      <ns0:priority>2</ns0:priority>
      <ns0:clusterMember>false</ns0:clusterMember>
    </ns0:construction>
  </ns0:adapterCluster>
</ns0:adapterClusterGroup>
</ns0:clusterGroup>
<ns0:deploymentPolicy>allowPartialDeploy</ns0:deploymentPolicy>
<ns0:encryptAlgorithm>AES128</ns0:encryptAlgorithm>
<ns0:debuggingMode>false</ns0:debuggingMode>
</ns0:bizSystemInfo>

```

3.5.1. Thread Pool

Set thread pools that can be used in the runtime engine. Detailed configurations for thread pools can be set commonly to all business systems or differently for each development runtime engine server.

The following shows a configuration example and a description of each item.

```

<ns0:threadPool>
  <ns0:id>bizSystemDefaultThreadPool</ns0:id>
  <ns0:name>bizSystemDefaultThreadPool</ns0:name>
  <ns0:queueFullPolicy>SystemError</ns0:queueFullPolicy>
  <ns0:useVipThread>false</ns0:useVipThread>
  <ns0:threadPriority>-1</ns0:threadPriority>
  <ns0:vipThreadPriority>-1</ns0:vipThreadPriority>
  <ns0:commonSetting>
    <ns0:queueSize>-1</ns0:queueSize>
    <ns0:min>60</ns0:min>
    <ns0:max>100</ns0:max>
    <ns0:keepAliveTime>300</ns0:keepAliveTime>
  </ns0:commonSetting>
</ns0:threadPool>

```

```

    <ns0:vipThreadCount>10</ns0:vipThreadCount>
  </ns0:commonSetting>
</ns0:threadPool>

```

Property	Description
id	Thread pool ID.
name	Thread pool name.
queueFullPolicy	Sets policies for full queues. <ul style="list-style-type: none"> – SystemError : system error handling – CallerThread : caller thread
useVipThread	Sets whether to use a specific number of VIP threads.
threadPriority	Sets the priority level of general threads. (Default value: 6)
vipThreadPriority	Sets the priority level of VIP threads. (Default value: 3)
queueSize	Sets the maximum size of job queues in thread pools. (Can be set for each server)
max	Maximum size of general thread pools. (Can be set for each server)
min	Minimum size of general thread pools. (Can be set for each server)
keepAliveTime	Default time for not cancelling an allocated thread. (Can be set for each server)
vipThreadCount	VIP thread count (fixed). (Can be set for each server)

3.5.2. System Logging

Set system logging information in the business system configuration information. A System log is an internal log of the AnyLink runtime engine server.

The user can set the default log level and the log level for each logger. The log level for each logger can be specified differently for each server.

The following predefined variable values can be used for fileLocation, which indicates where the log file will be stored.

```

<ns0:systemLogging>

<ns0:fileLocation>${server.home}/logs/AnyLink_rte_%d{1}%h{1}%m{30}.log</ns0:fileLocation>

  <ns0:logLevel>FINEST</ns0:logLevel>
  <ns0:logger>
    <ns0:name>com.tmax.anylink.serviceflow.level</ns0:name>
    <ns0:logLevel>INFO</ns0:logLevel>
  </ns0:logger>
</ns0:systemLogging>

```

```

</ns0:logger>
<ns0:logger>
  <ns0:name>com.tmax.anylink.runtime.level</ns0:name>
  <ns0:logLevel>FINE</ns0:logLevel>
</ns0:logger>
</ns0:systemLogging>

```

Property	Description
server.home	Location of the home directory for each server of the AnyLink runtime engine.
domain.home	Location of the home directory for each AnyLink domain.
install.root	Root directory location in which the AnyLink binary is installed.
server.name	Current AnyLink server name.
adminServer.name	Data integration server name of AnyLink.

3.5.3. Cluster

Set cluster configuration for each adapter of business systems that uses clusters. All adapters are enabled if no configuration is set.

The following is a configuration example and description of each item.

```

<ns0:clusterGroup>
  <ns0:adapterClusterGroup>
    <ns0:adapterCluster>
      <ns0:clusterId>TCP_ADT</ns0:clusterId>
      <ns0:activeCount>2</ns0:activeCount>
      <ns0:construction>
        <ns0:serverId>server4</ns0:serverId>
        <ns0:priority>0</ns0:priority>
        <ns0:clusterMember>true</ns0:clusterMember>
      </ns0:construction>
      <ns0:construction>
        <ns0:serverId>server5</ns0:serverId>
        <ns0:priority>1</ns0:priority>
        <ns0:clusterMember>true</ns0:clusterMember>
      </ns0:construction>
      <ns0:construction>
        <ns0:serverId>server6</ns0:serverId>
        <ns0:priority>2</ns0:priority>
        <ns0:clusterMember>false</ns0:clusterMember>
      </ns0:construction>
    </ns0:adapterCluster>
  </ns0:adapterClusterGroup>
</ns0:clusterGroup>

```

Property	Description
clusterId	Cluster ID.
activeCount	Maximum number of active nodes.
construction	Sets the following for each server node. <ul style="list-style-type: none"> – serverId : sets the server ID. – priority : sets the server priority. – clusterMember : option to include in a cluster.

3.5.4. Deployment Policy

By default, if a resource is not deployed to all servers, then the resource deployment will have failed. However, if some servers suffer from abnormal operation in a clustered environment, then deployment to just some servers can be performed.

The following shows a configuration example and a description of a property.

```
<ns0:deploymentPolicy>allowPartialDeploy</ns0:deploymentPolicy>
```

Property	Description
deploymentPolicy	Sets the deployment policy. <ul style="list-style-type: none"> – allOrNothing : Failed if not all are successfully deployed. – allowPartialDeploy : Allows deployment for alive servers. Automatically synchronized if an undeployed server is started.

3.5.5. Encryption Algorithm

Set the common encryption algorithm used in AnyLink.

The following shows a configuration example and a description of a property.

```
<ns0:encryptAlgorithm>AES128</ns0:encryptAlgorithm>
```

Property	Description
encryptAlgorithm	Sets the encryption algorithm. <ul style="list-style-type: none"> – AES128 : Encrypts by using AES128. No separate configuration is necessary.

Property	Description
	<ul style="list-style-type: none"> – AES256 : Encrypts by using AES256. Need to set configuration to allow JRE to use the encryption algorithm.

3.5.6. Debugging Mode

Set whether to allow flow debugging in the business system.

The following is a configuration example and a description of a property.

```
<ns0:debuggingMode>false</ns0:debuggingMode>
```

Property	Description
debuggingMode	<p>Sets the debugging mode.</p> <ul style="list-style-type: none"> – true : Allow debugging mode. Studio can use flow debugging for deployed resources. – false : Do not allow debugging mode. Cannot use flow debugging.

Chapter 4. Service Flow Engine

This chapter describes the basic elements and patterns of diagrams created using the service flow engine.

4.1. Service Flow Diagram

AnyLink provides functionalities for linking and orchestrating various adapters and services via the service flow engine. AnyLink connects incoming and outgoing services via a service flow to handle additional tasks or create new values.

AnyLink service flows use Business Process Modeling Notation (BPMN), which is the standard used for modelling business processes. Diagrams in the BPMN format are stored in the Service Flow Description Language (SFDL) format, which is an XML document format.

4.1.1. Basic Elements of Service Flow Diagrams

The following is a description of the basic elements of service flow diagrams.

- Service Flow

Called a process and consists of activities, events, and transitions.

- Activity

Called a task, and represents the tasks to be executed when a flow reaches the applicable position in the service flow.

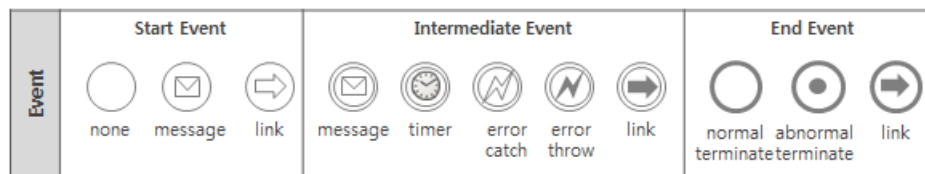
- Transition

An arrow indicating the order of a flow between activities or events.

- Event

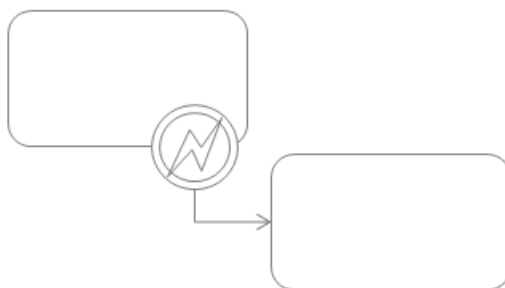
A special activity that indicates events such as messages, errors, and timeouts. Events consist of start events which do not have incoming transitions, intermediate events which have both incoming and outgoing transitions, and end events which do not have outgoing transitions.

[Figure 4.1] Events



Boundary events, which are a special form of intermediate events, are events that are positioned at the boundary of an activity or block, and if there is an event that occurs at the time a block or activity is activated, it will split off to the boundary event.

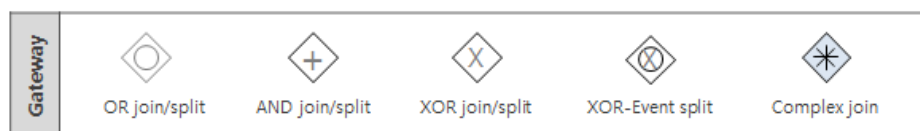
[Figure 4.2] Boundary Event



- Gateway

A special activity type used for controlling flows. It consists of a split gateway which splits off a single transition flow to multiple transition flows, and a join gateway which joins multiple transition flows into a single transition flow.

[Figure 4.3] Gateways



4.1.2. Service Flow Parallel Processing

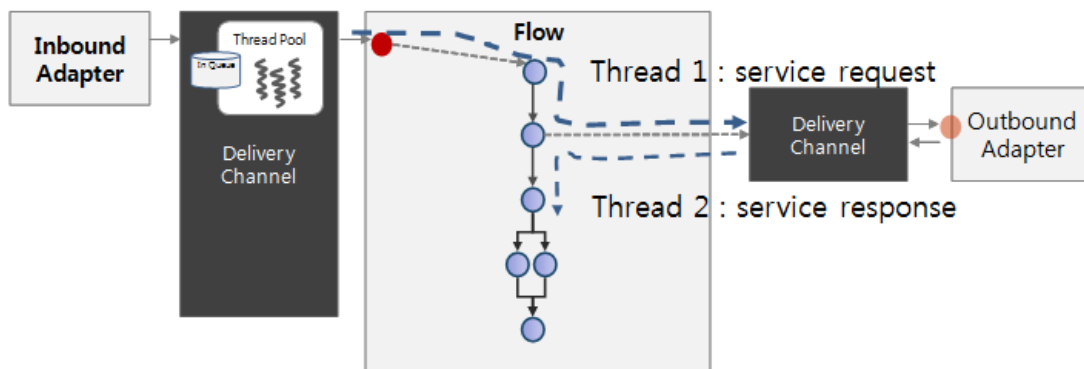
AnyLink service flows are executed in the service flow engine of the runtime engine server.

Typically, an instance of a service flow defined by a message delivered by an inbound adapter is created. A message or timer information sent to a service flow is called a trigger in a service flow.

Typically, a service flow generated by a single trigger is executed with a single thread. However, an SFDL that defines service flows can be split off in parallel or joined.

The AnyLink service flow engine can also execute a single service flow instance simultaneously with multiple threads according to the service flow diagram type. As well, the AnyLink service flow engine is designed in a way so that when a service flow is created and the flow is processed, the usage of threads (which are system resources) is maximized and multiple threads are simultaneously processed.

[Figure 4.4] Efficient Thread Processing Structure of Service Flow Engine without Wait Time



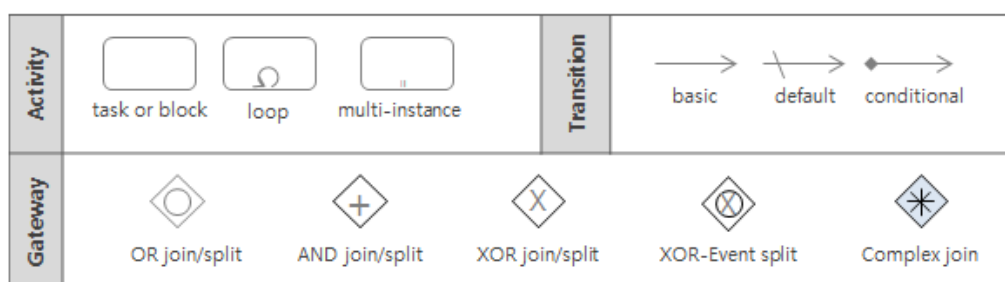
The service call thread (thread 1) does not wait for a response after sending a request. The service response processing thread (thread 2) is created in an adapter when a response is received.

4.1.3. Diagram Graphs and Block Structure

An AnyLink service flow is based on a graph structure in which activities are connected with arrows. The graph structure has the advantage of being intuitive and flexible because it can express flows in the way a person thinks.

Each activity, transition (represented by arrows), and gateway (which represents splitting off and joining of various flows) efficiently represents the service flow of such graph structure.

[Figure 4.5] Elements of a Graph Structure in a Service Flow - Activities



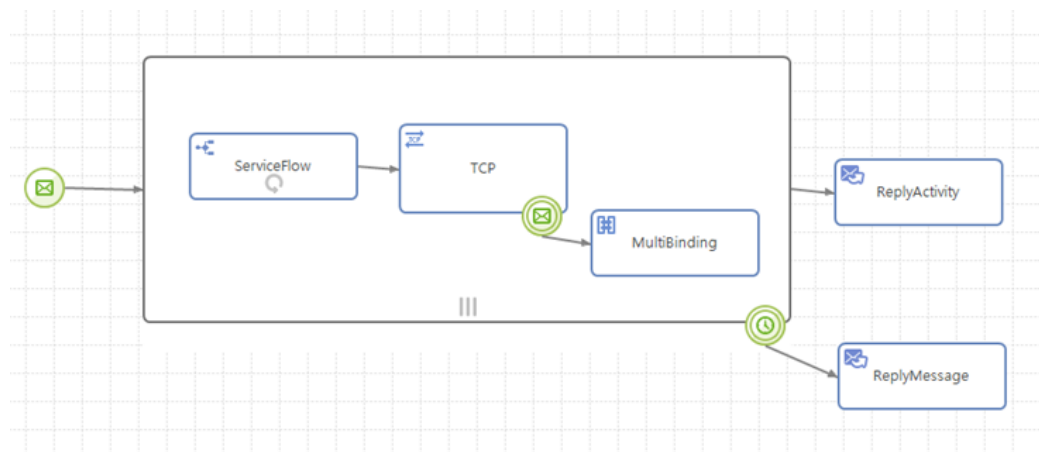
When handling events and exceptions that occur in a particular area, it is efficient to group them together and express them. Typically, a grouping of a particular flow is called a block.

In programming languages like Java, the basic execution unit is a block or function that groups together multiple statements. Errors that are generated in a particular block or function can be handled by a single error processing logic. In an AnyLink flow, the grouping of activities is called a subprocess or a block.

Events such as errors or timeouts that occur during the execution of an activity or transaction in a block move according to the error event or timer event attached to the block boundary. Typically, a good process consists of a graph structure and a block structure. An AnyLink service flow partially supports the block structure based on the graph structure.

In an AnyLink service flow diagram, boundary events that are attached at the boundary between a block activity and another block activity are examples of implementing the flow of the block structure.

[Figure 4.6] Block Structure Elements of a Service Flow - Block Activity



In addition to block activities, error handlers can be specified in service flows, which means errors generated when executing service flows can be handled. This can be thought of as a form of a block structure.

4.2. Basic Pattern

As shown in the previous AnyLink service flow diagram, flows are expressed by combining the graph structure and block structure based on the BPMN standard.

This section describes the commonly used service flow patterns.

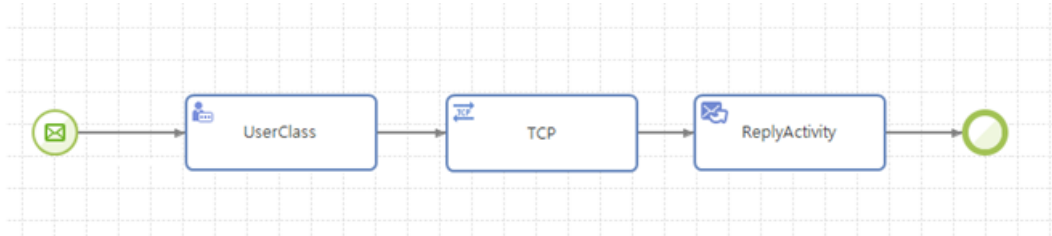
4.2.1. Basic Flow Pattern

The following describes the basic flow patterns.

- **Sequential Execution**

Sequential flows are the most basic flows, and an activity is executed after activities prior to it finish executing.

[Figure 4.7] Sequential Execution



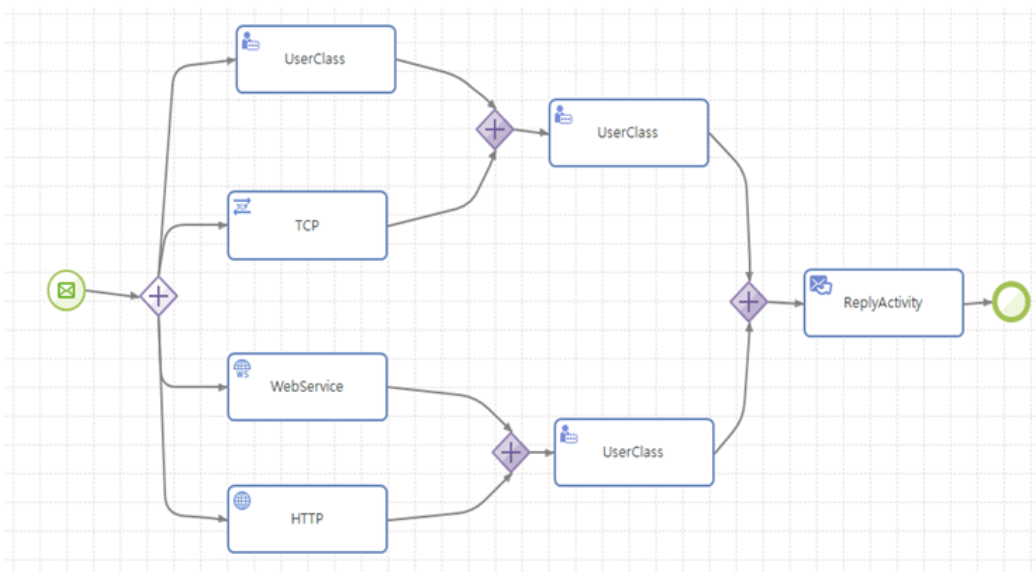
- **Parallel Split**

A single activity is split into multiple activities, and each of the split activities is executed independently of each other. Typically represented by the And Split gateway.

- **Synchronization**

Threads executed in parallel are synchronized and combined into a single flow. Typically represented by the And Join gateway.

[Figure 4.8] Parallel Split and Synchronization



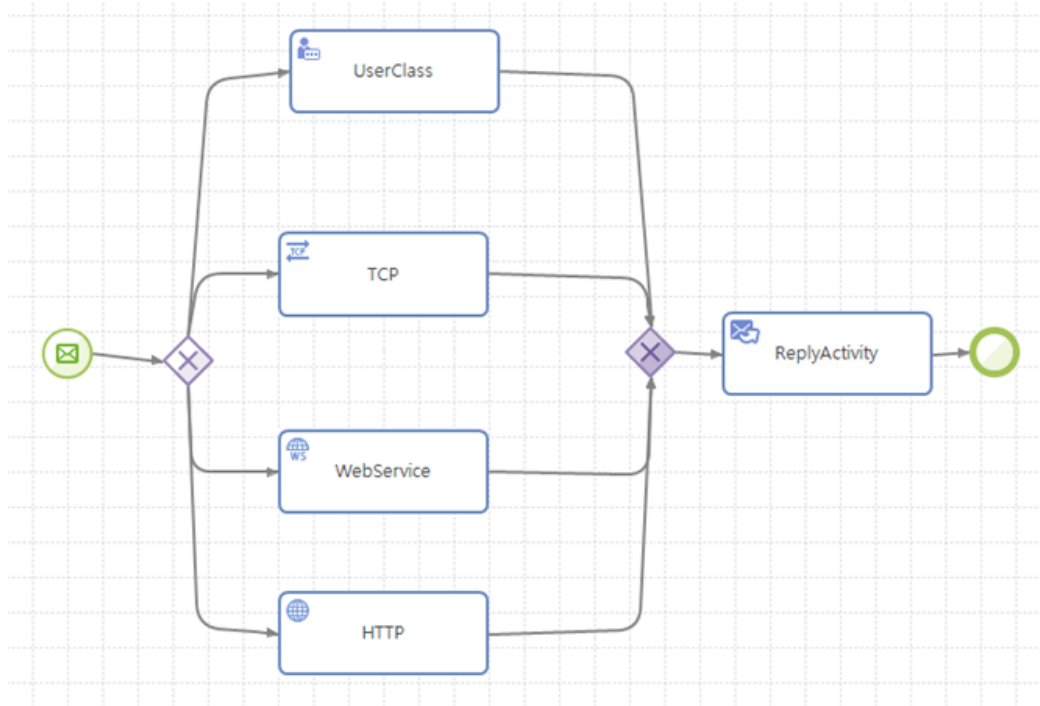
- **Exclusive Choice**

A single execution path among multiple execution paths is selected. Typically represented by the XOR Split gateway.

- **Simple Merge**

Only a single execution path among multiple execution paths is enabled for merging. Typically represented by the XOR Join gateway.

[Figure 4.9] Exclusive Choice and Simple Merge



4.2.2. More Split and Merge Patterns

The following describes more split and merge patterns.

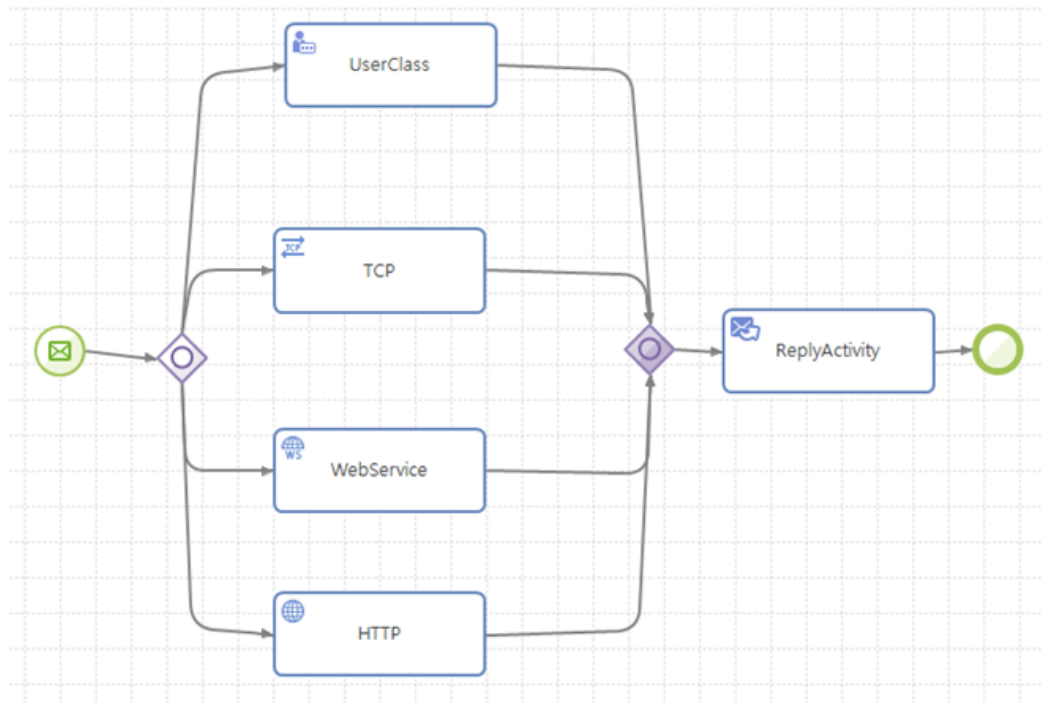
- **Multiple Choice**

Selects several execution paths among multiple execution paths and executes in parallel. Typically represented by the OR Split gateway.

- **Synchronizing Merge**

Joins while synchronizing multiple execution paths. Typically represented by the OR Join gateway.

[Figure 4.10] Multiple Choice and Synchronizing Merge

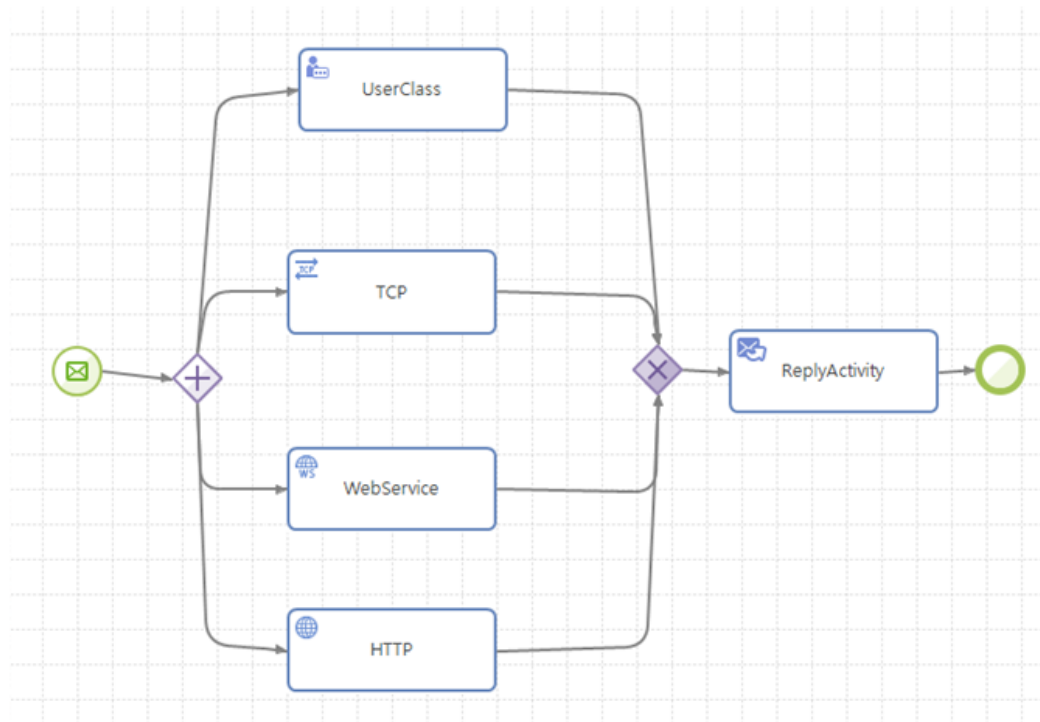


- **Multiple Merge**

Joins multiple execution paths into a single path without synchronization. Goes through a single path multiple times if multiple paths are running.

Runs as follows if merged with an XOR Join gateway after an And Split gateway.

[Figure 4.11] Join without Synchronization

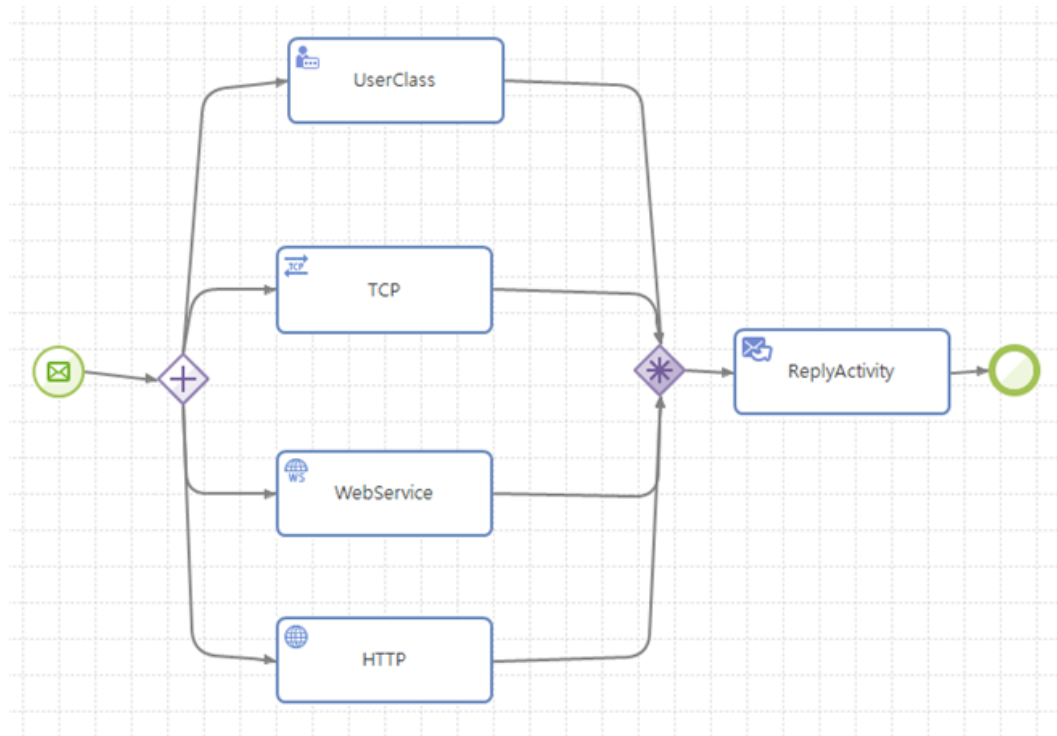


- **Discriminator**

Without synchronizing multiple execution paths, only the first arriving flow goes through continuously. If multiple paths are running, then only the first arriving thread goes through a single path.

If Complex Join gateway's flow condition is 1 when merging through a Complex Join gateway after an And Split gateway, the following occurs.

[Figure 4.12] Discriminator



- **N-out-of-M Join**

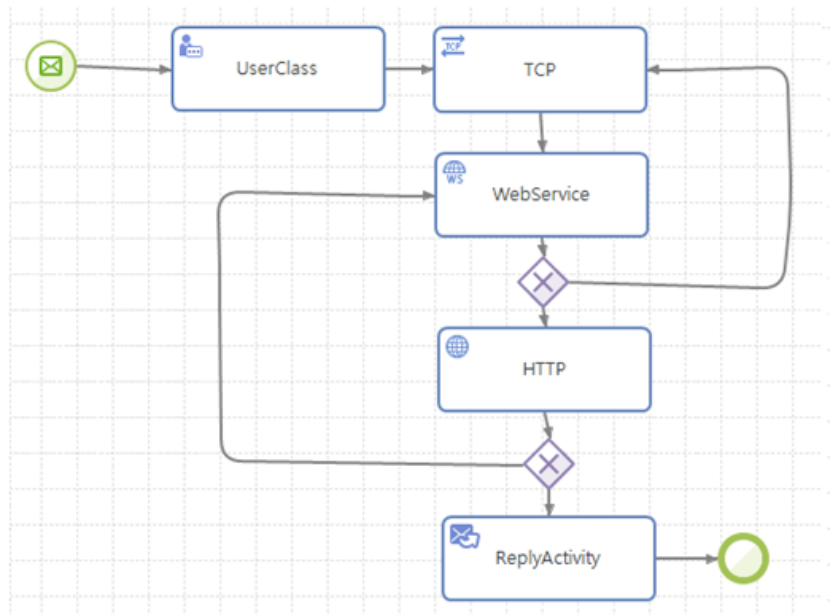
Synchronizes until the specified number of execution paths is reached, and then flows as one. Only one flow goes through for paths after this gateway.

If joining through a Complex Join gateway after an And Split gateway, the number of flows that come into the Complex Join gateway is N.

- **Arbitrary Cycle**

In AnyLink, repeat settings can be used for blocks. However, to repeat more freely, cycles can be set using a transition.

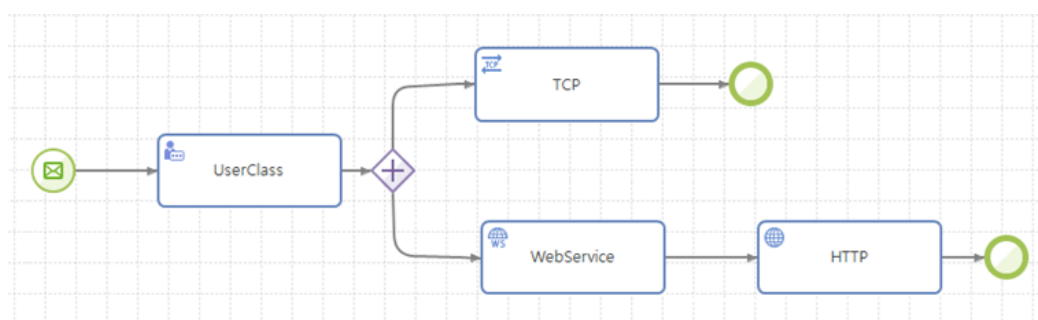
[Figure 4.13] Arbitrary Cycle



- **Implicit Termination**

If all running service flows are finished, they are terminated implicitly. For reference, they can be terminated explicitly by using a terminate event.

[Figure 4.14] Implicitly Terminating Service Flows



- **Multiple Instances**

Multiple activities are processed sequentially or in parallel with multiple instances. Multiple instances are created for each activity.

Multiple instances set the following.

Property	Description
Iteration Completion Condition	Specifies the Boolean condition that represents the completion condition or the number of instances.
Iteration Type	Two types exist: sequential and parallel. The sequential execution type is executed in the same method as a typical iteration statement, and the parallel execution type executes multiple instances at the same time. The parallel execution type is similar to the way multiple paths are executed through And Split.
Transition Method	<p>There are four transition methods.</p> <ul style="list-style-type: none"> – Transition occurs after all instances have been completed. Similar to the And Join method. – Transition occurs immediately if a single instance has been completed among multiple instances. – Transition occurs whenever each instance has been completed among multiple instances. – Transition occurs if a specified Boolean condition is met.

The following describes some key usage patterns related to multiple instances (MI).

– MI without Synchronization

This is for a case where multiple instances of an activity are created without synchronization, and a transition occurs every time an instance is completed.

– MI with a Prior Known Design Time Knowledge

This is for a case where the number of instances to be created for an activity to be executed in the MI method is known during design, and the instance count is designated as a constant in the iteration completion condition.

– MI with a Prior Known Runtime Knowledge

This is for a case where the number of instances to be created for an activity to be executed in the MI method can be known during the activity execution, and the instance count is designated as a variable in the iteration completion condition.

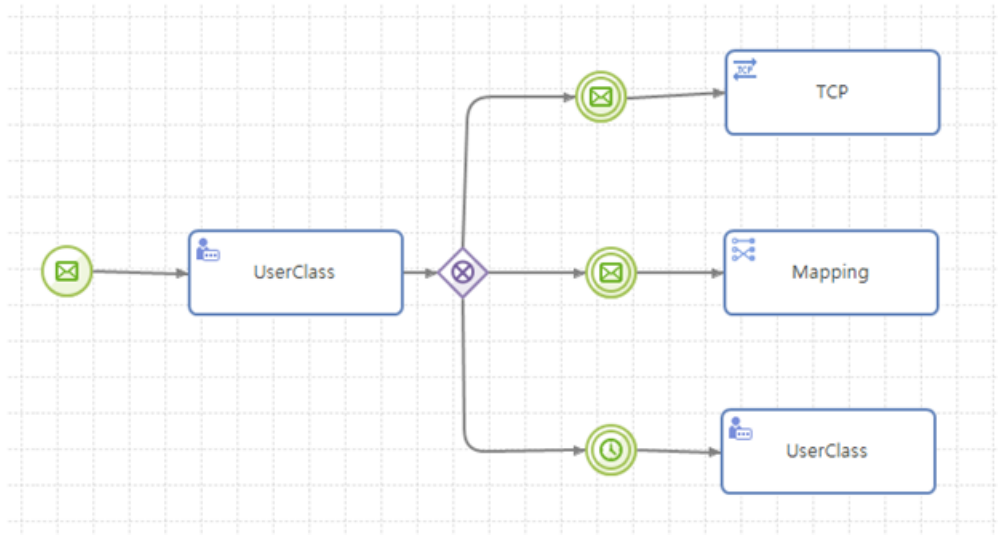
– MI without a Prior Runtime Knowledge

This is for a case where the number of instances to be created for an activity to be executed in the MI method cannot be known in advance even during execution. The iteration completion condition is specified by a Boolean condition.

- **Deferred Choice**

Selects a single path among multiple paths like the XOR gateway, but determines the path according to an event that occurs first among the queued events in multiple paths. Typically represented by the XOREvent gateway.

[Figure 4.15] Deferred Choice



4.3. Service Implementation Method and Service Flow Type

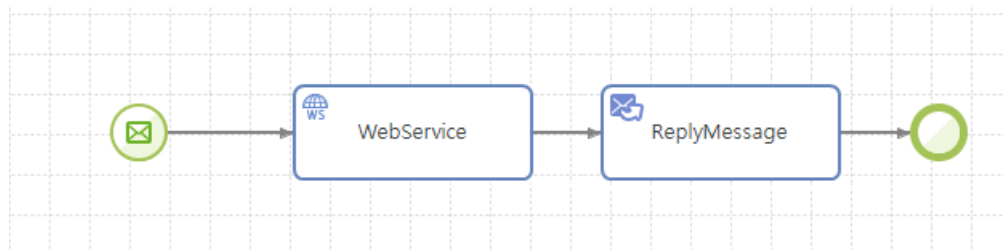
This section describes various service implementation methods and service flow types.

4.3.1. Proxy Service

A service model which integrates service endpoints so that existing or separately constructed services are serviced via AnyLink, and adds additional functions such as logging or security to existing services. For example, if an existing service is a web service, the WSDL of the service is imported to create a wrapper service.

A service flow is based on an incoming message event, outgoing service activity, and reply activity. Among them, activities for handling necessary logging and security are added.

[Figure 4.16] Service Flow that Implements Proxy Services

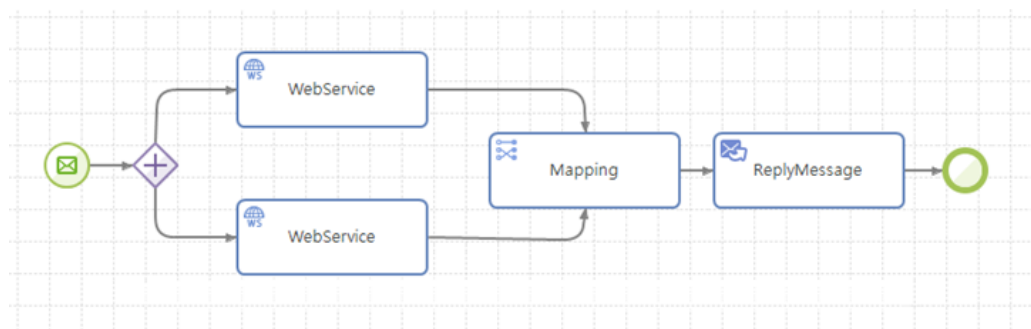


4.3.2. Composite Service

A service model in which AnyLink creates new services by joining various services that exist already or are separately constructed.

A service flow is based on an incoming message event, multiple outgoing service activities, and a reply activity. Among them, necessary logging, security, message conversion, and other activities are inserted.

[Figure 4.17] Service Flow that Implements Composite Services

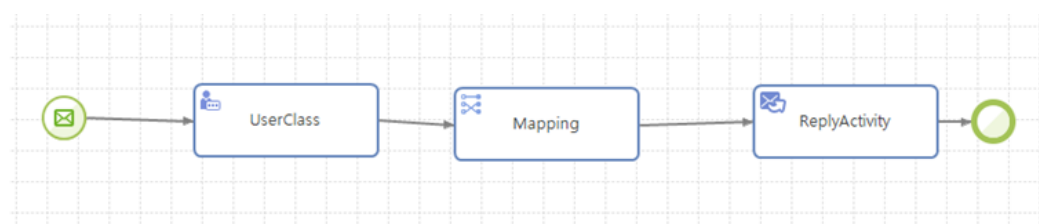


4.3.3. Service Implementation

AnyLink implements new services. The service logic is implemented mostly through user class activities, various adapter activities, and by splitting off.

A service flow is based on an incoming message event, a reply activity, and various adapter service activities and user class activities.

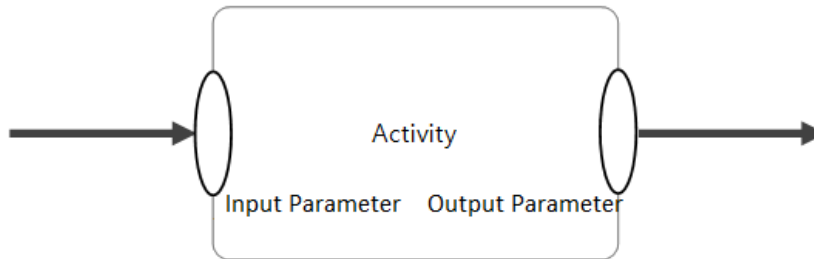
[Figure 4.18] Service Flow in Service Implementation Type



4.4. Service Flow Variable and Activity Parameter

The most basic form of an AnyLink service flow expresses a flow by connecting an activity and event with a transition. Here, the activity has parameters that correspond to input and output.

[Figure 4.19] Activity Input and Output



- An input parameter is a variable that an activity uses to read. The variable is declared in a block activity that includes processes or activities.
- An output parameter is a variable that an activity uses to write. The variable is declared in a block activity that includes processes or activities.

A service flow variable can be declared in a service flow (which is a process) or in a block activity that groups activities. The scope for process variables is all the activities, events, and transitions belonging to the process. The scope for block variables is all the activities, events, and transitions in the block activity.

4.5. Service Activity and Parameter

In an AnyLink service flow, calling another service is usually expressed as a call through an activity that represents the service. Services consist of **message events**, **adapter outbound rules**, and **multi binding router rules** of each service flow managed in the service flow engine.

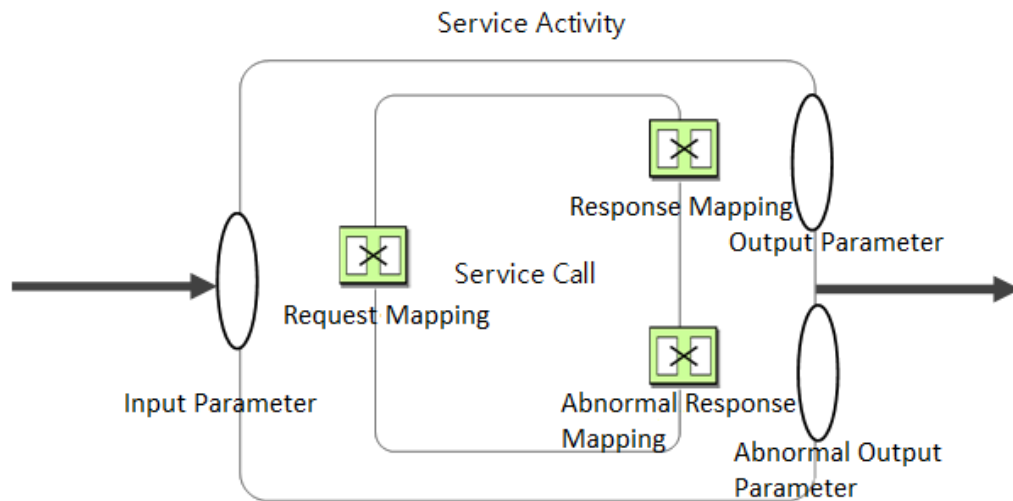
The act of calling another service flow in a service flow is called a **Service Flow Activity**. An adapter outbound rule call is represented by each adapter's activity (TCP, HTTP, Tmax, web service, etc.). As well, a multi binding activity is used for calling a multi binding router rule.

The characteristics of each service that calls service call activities may be different, but in the service flow perspective, they are operated the same as services called through a delivery channel. However, in the case of calling another service flow, the service flow engine internally calls itself, so it does not go through the delivery channel but is logically the same.

In an AnyLink service, request, response, and abnormal response messages are divided into input and output messages. A **service activity** is an activity that calls services, and can declare abnormal output parameters in addition to input and output parameters. In a typical service activity, an input parameter is

sent as a service's request message, and response and abnormal response messages are stored in output parameters and abnormal output parameters. If the input parameter and request message formats are different, data conversion can be specified by using the service activity's request mapping. As well, if the response message and output message formats are different, data can be converted by using response mapping, and if the abnormal response message and the abnormal output message formats are different, data can be converted by using abnormal response mapping.

[Figure 4.20] Service Activity Input



4.6. Mapping

Data conversion in an AnyLink service flow is done through mapping activities or service activities' request mapping, response mapping, and abnormal response mapping.

An AnyLink service flow can declare a process variable shared in a process. As well, a block activity can declare a block variable shared in a block.

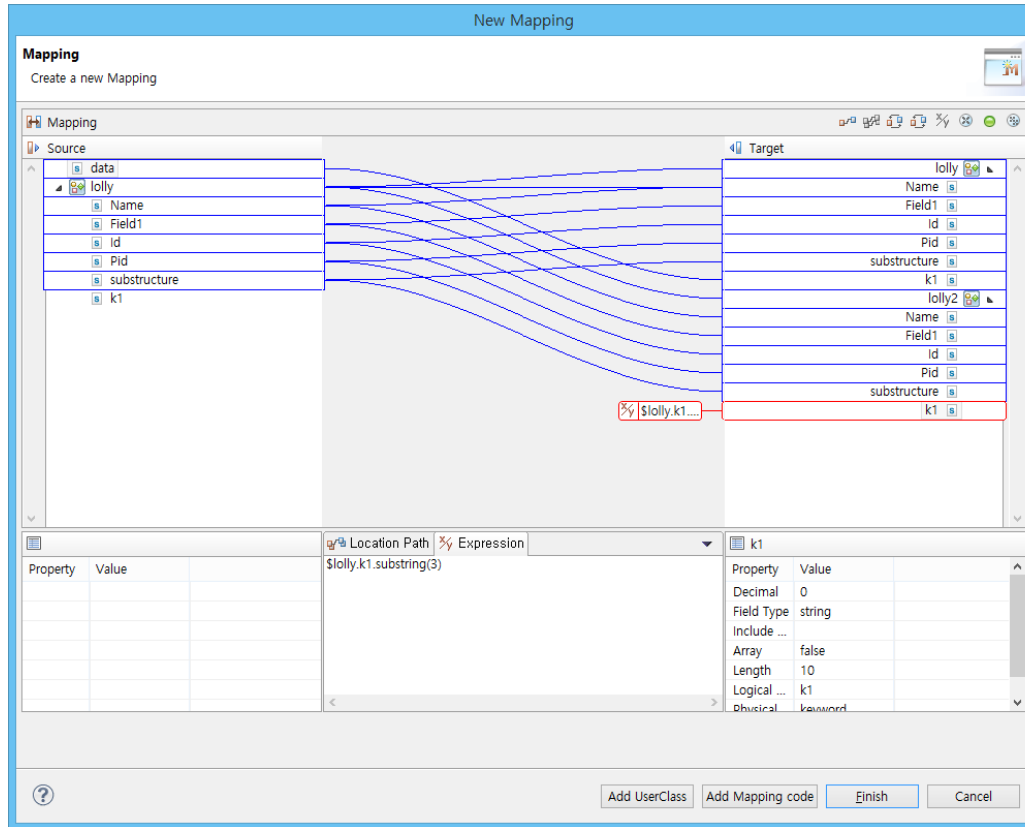
Supported variables are String, Float, Integer, DateTime, and Boolean. In addition, message data types (which are structured data types) are supported. Message data types used in an AnyLink service flow are represented as a class that indicates data in the Java programming language. They can be thought of as classes in the form of value objects or data transfer objects in the Java programming language.

In an AnyLink service flow, variables are internally converted to Java classes, and mapping is generated in Java code that converts and passes data between Java objects to improve execution performance. The most basic form of mapping is linking each field of the source message and target message. Hence, the most basic form is to assign a specific field value in the source message to a specific field in the target message.

In addition to the field of a message, an expression can be used as the source. Expressions that can be used in an AnyLink service flow will be described separately later. A mapping activity of an AnyLink service flow is an activity that directs message conversion between these process variables or block variables.

The source variable and target variable can be set, and the conversion rule between them can be specified using a diagram as follows.

[Figure 4.21] Mapping Editor



There are request mapping (for request data needed when calling the start event of an adapter rule or other service flow), response mapping (for changing the format of the results value), and abnormal response mapping (when an abnormal response message is sent). The request mapping maps the input parameter of a service activity into the request message format of a service (adapter rule or another service rule) to be called. The response mapping maps the response message of a service into the output parameter of a service activity. Like the response mapping, the abnormal response mapping maps the abnormal response message of a service into the abnormal output parameter of a service activity. A reply activity can specify the response mapping or abnormal response mapping.

4.7. Service Flow Expression

Expressions are used to hierarchically represent a specific value of a data object (variable) in statements for representing conditional branches of service flows, calculating correlation values, representing multi binding router conditions, etc.

By considering the performance of frequently called expressions, the AnyLink service flow engine creates the expressions in Java code and then compiles and deploys them.

4.7.1. Structured Expression

The most basic format of an expression is "variable name". "field name". For example, if there is a variable named "child" and if a field named "name" is defined in this variable, then the source can be expressed as child.name. If used as a source, child.name is converted to the same type as child.getName() when converted to Java code. This kind of expressions that represent structure is called structured expressions.

In mapping's each assignment operation, both the source and target can be specified as structured expressions. In the case of the source, various types of functions can be used in addition to structured expressions. Taking into consideration the characteristics of expressions generated as Java code, it is possible to use Java methods or operators in expressions of the mapping source. For example, if the name field of a variable named child is a string class in Java code, then the following expression can be used.

```
child.name.substring(3) + "_ChildName"
```

The above expression is converted in Java code as follows.

```
child.getName().substring(3) + "_ChildName"
```

4.7.2. Variable Expression

In the case of variable names, if there is no special notation, the beginning token of the structured expression is regarded as a variable name. Variables in a service flow are internally predefined and generated as code. It may be necessary to reference a service flow variable or a block variable during the execution of a service flow. The variables are represented by the "\$" notation. For example, if a variable named dataField1 is declared, it can be used as follows.

```
$dataField1.content
```

As previously described, expressions are created in Java code. If the exact type of a variable or field is not known, then the Java code cannot be generated correctly. For variables defined in a service flow, their information is extracted from their definitions and then used to assume their actual data types in order to generate the Java code. The data type of a variable can be explicitly specified by using "<>". For example, if the exact data type of a variable named dataField1 is the com.tmax.anylink.ContentContainer class, then it can be expressed as in the following. AnyLink internally converts expressions into Java code and compiles them, so if an exact data type is unknown, then an error will occur during deployment. In the case a data type cannot be assumed in AnyLink, compile errors can be avoided by explicitly specifying the data type.

```
$dataField1<com.tmax.anylink.ContentContainer>.content
```

The above expression is converted into the following Java code.

```
((com.tmax.anylink.ContentContainer) _varCtx.getVariable("dataField1")).getContent()
```

4.7.3. Mapping Expression

Use mappings in **mapping activities** to convert input/output data in service activities or to change variable values in service flows. In mapping, the source data can be represented by a separate expression instead of the input value. The expression at this time is slightly different from expressions such as the conditional statements or the correlation value calculation statements described above. The basic form of the expression used in a mapping's source is a structured expression. Hence, specify the field of the object value by using a period (.). In a hierarchical structure separated by a period (.), the first part of the mapping expression is the source or the target variable name. Unlike regular expressions, mapping expressions do not support variable names that begin with "\$". Instead, the beginning of a structured expression is the variable name. For example, input1 in the below example is the variable name.

```
input1.person.name
```

If the field object used in the source is null, and the child field object is hierarchically specified (that is, input1.person.name is specified when the input1.getPerson() value is null), then the object is created as an empty object so that no error is generated. In addition, if the field of the mapping is an array, an asterisk (*) can be attached after the field name used in the target and the field name used in the source to represent an array mapping. Unlike regular expressions, functions are expressed in the form starting with "@" in a mapping expression.

```
@substring(input1.person.name, 0, 4)
```

The following are the functions that can be used in a mapping expression.

Function	Description
@arraySize(arg1)	Returns the array size. – arg1 : structured expression.
@date(arg1?)	Returns the current date in the format indicated by the argument. (Default format: yyyy-MM-dd) – arg1 : Date format pattern string that can be used in Java's SimpleDateFormat.
@time(arg1)	Returns the current time in the format indicated by the argument. – arg1 : Date format pattern string that can be used in Java's SimpleDateFormat.
@strlen(arg1)	Returns the string length. – arg1 : Structured expression that represents a string object.

Function	Description
@trim(arg1)	Returns a string with no white space characters before and after the string. – arg1 : Structured expression that represents a string object.
@substring(arg1, arg2, arg3?)	Returns the substring of the string represented by the structured expression. – arg1 : Structured expression that represents a string. – arg2 : Starting offset. – arg3 : Ending offset.
@concat(arg1, arg2?, arg3?, ...)	Returns a string that combines multiple strings together. Each argument is a structured expression that represents a string, char, or numeric value.
@replace(arg1, arg2, arg3)	Returns a string that replaces a part of the string with another string. – arg1 : Structured expression that represents the original string. – arg2 : Substring to be replaced. – arg3 : Substring to replace with.
@dateformat(arg1, arg2, arg3)	Converts a string represented by a specific date format to another date format. – arg1 : Structured expression that represents a string expressed in a specific date format. – arg2 : Date format to be parsed. – arg3 : New date format.
@charAt(arg1, arg2)	Returns a character at a specific position in a string. – arg1 : Structured expression that represents a string. – arg2 : Position of the character.
@isNull(arg1)	Returns whether the result value of the structured expression is null. – arg1 : Structured expression.

Function	Description
@numberformat(arg1, arg2)	Converts the numeric value to a specific format. <ul style="list-style-type: none"> – arg1 : New format. – arg2 : Structured expression that represents the numeric value.
@urlEncode(arg1)	Returns a string URL-encoded with UTF8 charset. <ul style="list-style-type: none"> – arg1 : Structured expression that represents the string to be URL-encoded.
@getNull()	Returns the null object.
@equals(arg1, arg2)	Compares the equality of two objects. <ul style="list-style-type: none"> – arg1, arg2 : Structured expression.
@sequence(arg1, arg2, arg3)	Acquires a sequence. <ul style="list-style-type: none"> – arg1 : Represents the sequence ID. – arg2 : Represents the length of the sequence string. – arg3 : Selects whether to left pad the sequence string by the number of digits.

4.8. User Code and Handler

The AnyLink service flow provides a function in which users can write Java code to execute activities or execute additional code during error or at a specific point in time of the service flow and activity.

The following user code can be used in the service flow engine.

• User Class Activity

Activity that executes the Java code written by the user.

The Java code written by the user must inherit the DefaultUserActivity class of the com.tmax.anylink.api.serviceflow package.

```
// Main method of the user activity
void action(ActivityContext ctx) throws AnyLinkException;
```

• Process Handler

Handler class that is called at the start and end of the service flow.

The Java code written by the user must inherit the `DefaultProcessHandler` class of the `com.tmax.anylink.api.serviceflow` package.

```
// Called at the beginning of the service flow.
void started(ProcessContext ctx) throws AnyLinkException;

// Called at the end of the service flow.
void finished(ProcessContext ctx) throws AnyLinkException;

// Only defined internally. Not actually called.
void paused(ProcessContext ctx) throws AnyLinkException;

// Only defined internally. Not actually called.
void resumed(ProcessContext ctx) throws AnyLinkException;
```

- **Process Error Handler**

Handler class that is called when an error that cannot be handled during service flow execution occurs.

The Java code written by the user must inherit the `DefaultProcessErrorHandler` class of the `com.tmax.anylink.api.serviceflow` package.

```
// Called when an error occurs during the execution of a service flow in which the
// error handler is defined.
void handle(ProcessContext context, Throwable error) throws AnyLinkException;
```

- **Activity Handler**

Handler class that is called at the start and end (due to normal termination, cancellation, or an error) of the activity.

The Java code written by the user must inherit the `DefaultActivityHandler` class of the `com.tmax.anylink.api.serviceflow` package.

```
// Called at the start of the activity in which the handler is defined.
void started(ActivityContext ctx) throws AnyLinkException;

// Called at the end of the activity in which the handler is defined.
void finished(ActivityContext ctx) throws AnyLinkException;

// Called at the end (due to cancellation) of the activity in which the handler is
// defined.
void cancelled(ActivityContext ctx, String cause) throws AnyLinkException;

// Called at the end (due to an error) of the activity in which the handler is
// defined.
void errorOccurred(ActivityContext ctx, Throwable t) throws AnyLinkException;
```

- **Activity Error Handler**

Handler class that is called when an error that cannot be handled during activity execution occurs.

The Java code written by the user must inherit the `DefaultActivityErrorHandler` class of the `com.tmax.anylink.api.serviceflow` package.

```
// Called when an error occurs while an activity defined in the error handler is
executed.
void handle(ActivityContext context, Throwable error) throws AnyLinkException;
```

- **Error Code Mapper**

The service flow uses error code when generating or handling an error event. Hence, an error event written by the end event generates the specified error code's error events, and the boundary error event is responsible for handling errors in the activity according to the error code. The error code mapper converts Java exception objects that are generated during the execution of an activity to error code. The error code mapper class can be specified in the service flow.

The Java code written by the user must inherit the `DefaultErrorCodeMapper` of the `com.tmax.anylink.api.serviceflow` package.

```
// Implement this method which identifies an exception and determines its
corresponding error code.
String getErrorCode(Throwable throwable);
```

- **Service Activity Handler**

Handler class that is called before and after the call of the service.

The Java code written by the user must inherit the `DefaultServiceActivityHandler` class of the `com.tmax.anylink.api.serviceflow` package.

```
// Called just before the service call of the service activity in which the handler
is defined.
void beforeServiceCall(ActivityContext ctx, MessageContext mctx) throws
AnyLinkException;

// Called just after the service call of the service activity in which the handler
is defined.
void afterServiceCall(ActivityContext ctx, MessageContext mctx) throws
AnyLinkException;
```

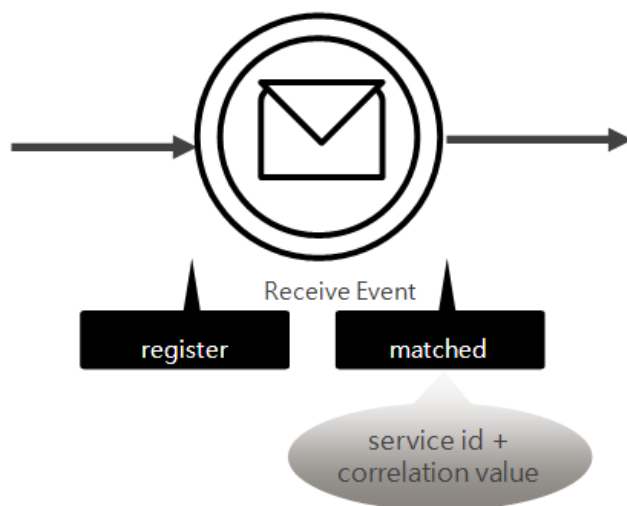
4.9. Correlation

A correlation refers to the association of independent objects. Correlation in AnyLink is used by adapters to find request messages and corresponding response messages. In AnyLink, a correlation does the following: when an event that represents a specific service is generated, an instance related to the event is found among the currently executed service flow instances, and the event is sent to the service flow instance. It associates a service flow instance with an event.

Typically, if an event generated in the service flow engine is a start event, it creates a new service flow instance. Events that use correlation are usually intermediate events where the service flow waits for an event to occur during execution, or boundary events.

In the service flow engine, a correlation event performs correlation between the service flow instance and service through the following two steps: the process of registering the expected correlation value, and a matching process for finding a matching correlation event instance when a corresponding service occurs.

[Figure 4.22] Two Steps of Service Correlation



When a service flow reaches an intermediate event using correlation through a transition, the AnyLink service flow engine internally calculates the value for correlation and registers it in the correlation information table.

If a message is sent to the service flow engine via the AnyLink delivery channel, it is handled by the service flow engine in the following order.

1. If the service is a service that corresponds to the start message event of the service flow, a new service flow instance is created.
2. If the service is a service that corresponds to the message event that requires correlation, it attempts correlation matching. If a message event that matches the correlation value is found, then the message

is sent to the service flow instance in which the event belongs to so that the flow instance transitions to the next step.

3. If correlation matching is not successful, it finds a message event waiting for the corresponding service without correlation, and delivers the service flow instance method in which the event belongs to so that the flow instance transitions to the next step.
4. If none of the message events waiting for the service are registered, then it is stored in the early arrived message map under the assumption that the message event has reached before the message event is registered in the service flow engine. The period of time to wait for a message arriving before the event registration is three seconds by default.

Correlation matching occurs in the AnyLink service flow engine when Service IDs (composed of a combination of the service flow ID and the corresponding message event ID) are the same and the calculated correlation value is the same as the correlation value registered by the service flow instance. For the correlation matching, intermediate events in the service flow can specify registration and matching expressions.

If you need correlation between different service flows or message events, you can use a multi-binding router. When the routing method of the multi-binding router is selected as the service flow correlation, correlation value matching can be performed between different services.

Chapter 5. Multi-binding Router

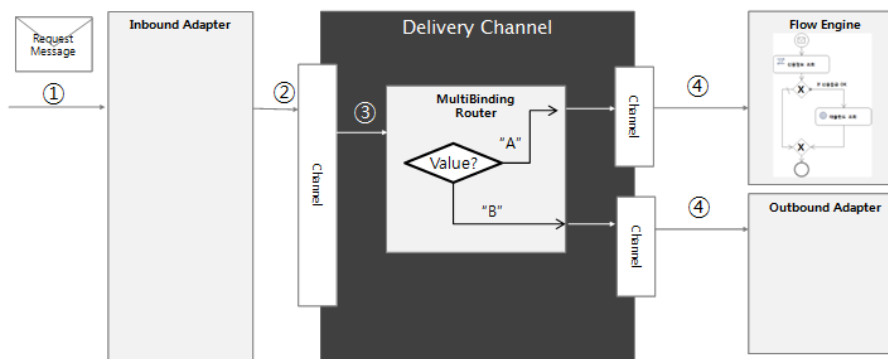
This chapter describes how to split services in the multi-binding router and the router's rules.

5.1. Overview

Multi-binding router provides the functionality for selectively splitting or multi-casting AnyLink services. Services that the multi-binding router can split or multi-cast are AnyLink internal services such as a service flow's received message events, adapter outbound rules, and other multi-binding router rules.

A multi-binding router rule is assumed to be an AnyLink service. Hence, it can call a multi-binding rule by using the adapter's parsing rule, and can also be called through multi-binding activities among the service activities of the service flow or other multi-binding rules.

[Figure 5.1] Splitting a Service According to the Multi-binding Router Rule



5.2. Multi-binding Rule

A multi-binding rule consists of the following elements.

- Request, response, and abnormal response messages that call multi-binding rules as services.
- Information about target services to be called according to routing rules.
- Routing methods and parameters by service.

The `com.tmax.anylink.api.multibinding.DefaultRoutingHandler` class declares the following methods.

```
String route(MessageContext ctx) throws AnyLinkException;
```

The routing methods and parameters used in the multi-binding router are as follows.

Routing Method	Method
Value	Performed according to the value of a user-defined variable.
RoundRobin	Round robin. Services are routed sequentially.
WeightBased	Weighted round robin.
Multicast	Can be used only for one-way services. Services are sent to all registered entries.
TimeRange	Calls services that correspond to the specified time period.
FlowCorrelation	Service flow's correlation method. Uses value matching to route to the service item that is matched first. Here, the target service must be a service flow message event that supports correlation.
Handler	User handler class.

Note

For more information about each routing method, refer to the relevant section.

5.2.1. Value

Performed according to the value of a user-defined variable. Routes to the service item that matches the value of calculating an expression. The request message can be used as a variable called input in the expression.

– Format

```
$input.field1 (variable name)
```

– Example

```
1, 2, svc1
```

5.2.2. WeightBased

Weighted round robin.

– Format

```
weight. integer value
```

5.2.3. TimeRange

Calls services that correspond to the specified time period. If there is no service item for the matching time period, then routes to the default service item.

- Format

```
HH:MM - HH:MM
```

If the succeeding time is greater than the preceding time and the current time is in the middle, routing is performed. If the preceding time is greater than the succeeding time, routing is performed only when the current time is less than the succeeding time or greater than the preceding time.

- Example

```
09:30-15:10 or default
```

5.2.4. Handler

User handler class. Routes to the service item that matches the result value of calling a user handler class implemented by inheriting the DefaultRoutingHandler class.

- Format

```
telco.binding.TaxRRBinding (class name)
```

- Example

```
a, b
```


Appendix A. Server APIs

This appendix describes AnyLink user APIs, especially related to service flows and the multi-binding router.

Note

Most of APIs described in this appendix are defined along with a default implementation user class whose name starts with 'Default'. It is recommended to implement a user interface by inheriting the class.

A.1. com.tmax.anylink.api Package

The MessageContext interface which is used commonly in various packages is defined.

<MessageContext>

```
package com.tmax.anylink.api;

/**
 * Standard interface that represents a message.
 */
public interface MessageContext {
    /**
     * Returns the data of the @return message.
     */
    Object getContent();

    /**
     * Returns the property value of the @return message.
     */
    Object getProperty(String propertyName);

    /**
     * Specifies the message's property value.
     *
     * @param propertyName, property name
     * @param propertyValue, property value
     */
    void setProperty(String propertyName, Object propertyValue);
}
```

A.2. com.tmax.anylink.api.serviceflow Package

Defines interfaces for user classes and handlers used in service flows' processes or activities, and their default implementation classes. For example, the interface for a user activity is defined as the `UserActivity` interface, and the default implementation class is defined as the `DefaultUserActivity` abstract class.

Interface	Description
ActivityContext	Represents the execution context of an activity.
ActivityErrorHandler	Handler interface that can handle activity errors.
ActivityHandler	AnyLink activity handler interface.
BlockContext	Represents the execution context of a block activity including the currently executed activity.
ErrorCodeMapper	Converts internal errors into error code that can be recognized by the AnyLink service flow.
UserMapping	Allows the user to write data conversion code in the AnyLink service flow.
ProcessContext	Represents the execution context of the currently executed service flow.
ProcessHandler	AnyLink process handler interface.
ServiceActivityHandler	AnyLink service activity handler interface.
UserActivity	Represents the user activity.
VariableContext	Defines the interface that corresponds to the service flow variable.

Note

It is recommended to implement a user interface by inheriting a default abstract class.

A.2.1. ActivityContext

Represents the execution context of an activity.

```
package com.tmax.anylink.api.serviceflow;

/**
 * Interface that represents the execution context of an activity
 */

public interface ActivityContext {
    /**
     * @return returns the activity ID.
     */
    String getActivityId();
}
```

```

    /**
     * @return returns the variable context with the specified name as seen in this
    activity scope.
     */
    VariableContext getVariable(String variableName);

    /**
     * @return returns the context of the service flow that contains this activity.
     */
    ProcessContext getProcessContext();

    /**
     * @return returns the context of the block activity including this activity.
     * If the block activity is not enclosed, null is returned.
     */
    BlockContext getBlockContext();
}

```

A.2.2. ActivityErrorHandler

Handler interface that can handle activity errors.

```

package com.tmax.anylink.api.serviceflow;

import com.tmax.anylink.common.AnyLinkException;

/**
 * Handler interface used to handle activity errors
 */

public interface ActivityErrorHandler {
    /**
     * Method called when an error occurs during the execution of an activity in
    which the handler is defined.
     */
    void handle(ActivityContext context, Throwable error) throws AnyLinkException;
}

```

A.2.3. ActivityHandler

AnyLink activity handler interface.

```

package com.tmax.anylink.api.serviceflow;

import com.tmax.anylink.common.AnyLinkException;

/**
 * AnyLink activity handler interface.
 * When an object implementing this interface is registered as a handler for the
 * activity,
 * corresponding methods are called according to the life cycle time of the activity
 * instance.
 */

public interface ActivityHandler {
    /**
     * Method called at the start of the activity
     */
    void started(ActivityContext ctx) throws AnyLinkException;

    /**
     * Method called at normal end of the activity
     */
    void finished(ActivityContext ctx) throws AnyLinkException;

    /**
     * Method called when an activity is canceled without being terminated.
     * An activity can be canceled for a number of reasons, usually canceled
     * by another activity or event in the service flow.
     */
    void cancelled(ActivityContext ctx, String cause) throws AnyLinkException;

    /**
     * Method called when the activity is terminated by an error
     */
    void errorOccurred(ActivityContext ctx, Throwable t) throws AnyLinkException;
}

```

A.2.4. BlockContext

Represents the execution context of a block activity including the currently executed activity.

```

package com.tmax.anylink.api.serviceflow;

/**
 * Interface representing the execution context of a block activity including the
 * currently executed activity.

```



```

*/

public interface BlockContext {
    /**
     * @return returns the variable context with the specified name as seen in this
     block scope.
     */
    VariableContext getVariable(String variableName);

    /**
     * @return returns the execution context of the parent block activity including
     the block activity.
     * If there is no parent block activity, null is returned.
     */
    BlockContext getParentBlockContext();

    /**
     * @return returns the context of the service flow that contains the block
     activity.
     */
    ProcessContext getProcessContext();
}

```

A.2.5. ErrorCodeMapper

Converts internal errors into error code that can be recognized by the AnyLink service flow.

```

package com.tmax.anylink.api.serviceflow;

/**
 * Interface for converting internal errors into error code that can be recognized
 by the AnyLink service flow.
 * If the AnyLink service flow does not need to catch all exceptions, it needs to
 know the error code to handle the error.
 * This interface converts the generated Java objects into error codes so that error
 events in the service flow can catch them.
 */

public interface ErrorCodeMapper {
    /**
     * @param throwable, the generated Java exception object
     * @return returns the error code value to be handled by the error event of the
     service flow.
     */
    String getErrorCode(Throwable throwable);
}

```

A.2.6. UserMapping

Allows the user to write data conversion code in the AnyLink service flow.

```
package com.tmax.anylink.api.serviceflow;

/**
 * Interface that allows users to write data conversion code in the AnyLink service
 * flow.
 * Used for request, response, abnormal response, and custom log mappings by using
 * user-written code.
 */

public interface UserMapping {
    /**
     * User-written mapping method
     *
     * @param ctx, the current activity context
     * @param inputParams, input parameters are sent via this variable.
     * @return, the mapped result data. The number of result parameters and the array
     * size must be the same.
     */
    Object[] mapping(ActivityContext ctx, Object[] inputParams) throws
AnyLinkException;
}
```

A.2.7. ProcessContext

Represents the execution context of the currently executed service flow.

```
package com.tmax.anylink.api.serviceflow;

import com.tmax.anylink.logging.Logger;

/**
 * Interface that represents the execution context of the currently executing service
 * flow
 */

public interface ProcessContext {
    /**
     * @return returns the service flow ID.
     */
    String getProcessId();
}
```

```

/**
 * @return returns the instance ID of the service flow.
 */
String getProcessInstanceId();

/**
 * @return returns the system logger object.
 */
Logger getUserLogger();

/**
 * @return returns the variable information of the specified name in the scope
of this service flow.
 */
VariableContext getVariable(String variableName);

/**
 * Returns the AnyLink execution environment variable value.
 * server.name, cluster.name, bizsystem.id are the currently supported environment
variable keys.
 *
 * @param, key environment variable key
 * @return, environment variable value that corresponds to the key
 */
String getEnv(String key);
}

```

A.2.8. ProcessHandler

AnyLink process handler interface.

```

package com.tmax.anylink.api.serviceflow;

import com.tmax.anylink.common.AnyLinkException;

/**
 * AnyLink process handler interface.
 * When an object implementing this interface is registered as a handler of the
corresponding service flow,
 * the corresponding methods are called according to the lifecycle of the service
flow.
 */

public interface ProcessHandler {

/**
 * Method called when the service flow is started.

```

```

    */
    void started(ProcessContext ctx) throws AnyLinkException;

    /**
     * Method called when the service flow is finished.
     */
    void finished(ProcessContext ctx) throws AnyLinkException;

    /**
     * Method called when the service flow is paused. Currently, this method is not
     used.
     */
    void paused(ProcessContext ctx) throws AnyLinkException;

    /**
     * Method called when the paused service flow is resumed. Currently, this method
     is not used.
     */
    void resumed(ProcessContext ctx) throws AnyLinkException;
}

```

A.2.9. ServiceActivityHandler

AnyLink service activity handler interface.

```

package com.tmax.anylink.api.serviceflow;

import com.tmax.anylink.api.MessageContext;
import com.tmax.anylink.common.AnyLinkException;

/**
 * AnyLink service activity handler interface.
 * This interface defines additional methods for each point in time in the service
 call.
 */

public interface ServiceActivityHandler extends ActivityHandler {
    /**
     * Method called just before calling the service.
     */
    void beforeServiceCall(ActivityContext ctx, MessageContext context) throws
AnyLinkException;

    /**
     * Method called immediately after calling the service.
     */
}

```

```

        void afterServiceCall(ActivityContext ctx, MessageContext context) throws
AnyLinkException;
    }

```

A.2.10. UserActivity

Represents the user activity.

```

package com.tmax.anylink.api.serviceflow;

import com.tmax.anylink.common.AnyLinkException;

/**
 * Interface that represents the user activity.
 * To create a user class activity, users need to create a class that inherits the
DefaultUserActivity abstract class that implements this interface.
 */

public interface UserActivity {
    /**
     * Main method that the user activity class should implement.
     */
    void action(ActivityContext ctx) throws AnyLinkException;
}

```

A.2.11. VariableContext

Defines the interface that corresponds to the service flow variable.

```

package com.tmax.anylink.api.serviceflow;

import com.tmax.anylink.api.MessageContext;

/**
 * Defines the interface that corresponds to the service flow variable.
 */

public interface VariableContext extends MessageContext {
    /**
     * Specify the variable data.
     *
     * @param content, data to be specified in the variable
     */
}

```

```
void setContent(Object content);  
}
```

A.3. com.tmax.anylink.api.multibinding Package

Handler interfaces used by multi-binding routers are defined.

RoutingHandler

```
package com.tmax.anylink.api.multibinding;  
  
import com.tmax.anylink.api.MessageContext;  
import com.tmax.anylink.common.AnyLinkException;  
  
/**  
 * Multi-binding router's handler interface.  
 * This handler class can be used when the multi-binding rule's routing method is  
 * specified as "Handler".  
 */  
  
public interface RoutingHandler {  
    /**  
     * @return returns the routing entry name.  
     * To process the entry, the entry name must match at least one of entry values  
     * in the router rule.  
     */  
    String route(MessageContext ctx) throws AnyLinkException;  
}
```

A.4. Default Abstract Class List

List of abstract classes that implement each of the interfaces mentioned above.

- com.tmax.anylink.api.serviceflow.DefaultUserActivity
- com.tmax.anylink.api.serviceflow.DefaultActivityHandler
- com.tmax.anylink.api.serviceflow.DefaultProcessHandler
- com.tmax.anylink.api.serviceflow.DefaultActivityErrorHandler
- com.tmax.anylink.api.serviceflow.DefaultProcessErrorHandler

- `com.tmax.anylink.api.serviceflow.DefaultErrorCodeMapper`
- `com.tmax.anylink.api.multibinding.DefaultRoutingHandler`

Note

It is recommended to implement an interface by inheriting the default abstract class, for future extensibility.

Index

A

- Activity Error Handler, 52
- Activity Handler, 51
- Activity), 31
- ActivityContext, 60
- ActivityErrorHandler, 60, 61
- ActivityHandler, 60, 61
- Adapter, 1
 - Inbound Rule, 8
 - Outbound Rule, 9
- Adapter Resources, 23
 - Adapter, 24
 - Endpoints and Endpoint Groups, 24
- Arbitrary Cycle, 40

B

- Basic Pattern, 34
- Block, 34
- BlockContext, 60, 62
- BPMN, 7, 31
- Business Resources, 17
 - Transactions and Transaction Groups, 17
- Business System Configuration Information
 - Cluster, 28
 - Debugging Mode, 30
 - Deployment Policy, 29
 - Encryption Algorithm, 29
 - System Logging, 27
 - Thread Pool, 26

C

- com.tmax.anylink.api, 59
 - MessageContext, 59
- com.tmax.anylink.api.multibinding
 - RoutingHandler, 68
- com.tmax.anylink.api.serviceflow

- ActivityContext, 60
- ActivityErrorHandler, 60, 61
- ActivityHandler, 60, 61
- BlockContext, 60, 62
- ErrorCodeMapper, 60, 63
- ProcessContext, 60, 64
- ProcessHandler, 60, 65
- ServiceActivityHandler, 60, 66
- UserActivity, 60, 67
- UserMapping, 60, 64
- VariableContext, 60, 67

- Composite Service, 3, 43
- Configuring Business System Information, 24
- Correlation, 53

D

- Data Integration Server (DIS), 1
- Data Transfer Object, 45
- Deferred Choice, 42
- Delivery Channel, 9, 11

E

- Engine Architecture, 10
- Environment Configurations for Transactions and Transaction Groups, 21
 - Environment Configurations for Transactions and Transaction Groups (XML document files with the .bizcfg extension), 21
- Error Code Mapper, 52
- ErrorCodeMapper, 60, 63
- Event, 31
- Exclusive Choice, 35
- Expression, 46

F

- FlowCorrelation, 56

G

- Gateway, 32

H

- Handler, 56, 57

I

Implicit Termination, 40
Inbound Rule, 8
Input Parameter, 44

J

Java Class Loader and Transaction Tree Class Loading Method, 21

L

Libraries, 23

M

Mapping, 45
Mapping Expression, 48
MessageContext, 59
Messaging Pattern, 11
 Oneway, 12
 Oneway-ACK, 12
 Request-Response (2-way), 12
 Request-Response-ACK (3-way), 12
 Test-Message, 12
MI with a Prior Known Design Time Knowledge, 41
MI with a Prior Known Runtime Knowledge, 41
MI without a Prior Runtime Knowledge, 41
MI without Synchronization, 41
Multi-binding Router, 1, 9, 55
 Multi-binding Rule, 9
 Outbound Rule, 9
 Receive Message Events, 9
Multi-binding Router Method
 FlowCorrelation, 56
 Handler, 56, 57
 Multicast, 56
 RoundRobin, 56
 TimeRange, 56, 57
 Value, 56
 WeightBased, 56
Multi-binding Rule, 55
Multicast, 56
Multiple Choice, 36
Multiple Instances, 40

Iteration Completion Condition, 41
Iteration Type, 41
Transition Method, 41
Multiple Merge, 38, 39

N

N-out-of-M Join, 39

O

Outbound Rule, 9
Output Parameter, 44

P

Parallel Split, 35
Parsing Rule, 9
Process Error Handler, 51
Process Handler, 50
ProcessContext, 60, 64
ProcessHandler, 60, 65
Protocol Adapter, 8
Proxy Service, 3, 42

R

Reliable Messaging, 12
Reliable Messaging and XA Messaging, 12
Resource Manager, 9
RoundRobin, 56
RoutingHandler, 68
Runtime Engine (RTE) server, 1
Runtime Engine Error Processing, 13
Runtime Engine Server
 Multi-binding Router, 1
 Service Flow Engine, 1
Runtime engine server
 Adapter, 1
Runtime Engine Service, 13
 Business Class Loader, 13
 Deploy Manager, 13
 Resource Manager, 13
 System Resource Manager, 13
Runtime Engine Service Container (RTE Service Container), 10

S

- Sequential Execution, 35
- Server Architecture, 9
- Server Component
 - Adapter, 8
- Server Components, 7
 - Service Flow Engine, 7
- Service, 9
- Service Activity, 44
- Service Activity and Parameter, 44
- Service Activity Handler, 52
- Service Flow, 31
- Service Flow Activity, 44
- Service Flow Diagram, 31
- Service Flow Engine, 1
- Service Flow Expression, 46
- Service Flow Parallel Processing, 32, 33
- Service Flow Variable, 44
- Service Implementation, 43
- Service Implementation Method and Service Flow Type, 42
- Service Orchestration, 2
- ServiceActivityHandler, 60, 66
- SFDL, 31
- Simple Merge, 36
- Structured Expression, 47
- Subprocess, 34
- Symbolic Links and Java Class References, 19
- Synchronization, 35
- Synchronizing Merge, 37

T

- The Basic Hierarchy of a Java Class Loader, 22
- TimeRange, 56, 57
- Transaction Propagation, 13
- Transaction Tree and Message Parsing, 18
- Transactions and Transaction Groups
 - Adapter Outbound Rule (XML document files with a .orule extension), 19
 - Definition Files for Transactions and Transaction Groups (XML Document Files with the .biztx Extension), 17

- Expression (XML Document Files with the .expr Extension), 20
- Message (XML Document File with the .Umsg Extension), 20
- Message Mapping (XML Document File with the .Map Extension), 20
- Multi-binding Rule (XML Document Fields with the .mbind extension), 20
- Parsing Rule, 19
- Service Flow Definition File (XML Document Files with the .sfdl Extension)), 19
- Transition, 31
- Trigger, 32

U

- User Class Activity, 50
- User Code and Handler, 50
- UserActivity, 60, 67
- UserMapping, 60, 64

V

- Value, 56
- Value Object, 45
- Variable Expression, 47
- VariableContext, 60, 67

W

- WeightBased, 56
- Wrapper Service, 3

X

- XA Messaging, 12

