

BX Framework

BXCM 개발자 가이드 - 개발표준

v2.0

BX Framework: BXCM 개발자 가이드 - 개발표준

Publication date 2023-12-06

Copyright © 2023 BankwareGlobal Co., Ltd. All Right Reserved.

Restricted Rights Legend

All BankwareGlobal Software and documents are protected by copyright laws and the Protection Act of Computer Programs, and international convention. BankwareGlobal software and documents are made available under the terms of the BankwareGlobal 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 BankwareGlobal Co., Ltd.

이 소프트웨어 사용설명서의 내용과 프로그램은 저작권법, 컴퓨터프로그램보호법 및 국제 조약에 의해서 보호받고 있습니다. 사용설명서의 내용과 여기에 설명된 프로그램은 BankwareGlobal Co., Ltd.와의 사용권 계약 하에서만 사용이 가능하며, 사용권 계약을 준수하는 경우에만 사용 또는 복제할 수 있습니다. 이 사용설명서의 전부 또는 일부분을 BankwareGlobal의 사전 서면 동의 없이 전자, 기계, 녹음 등의 수단을 사용하여 전송, 복제, 배포, 2차적 저작물 작성 등의 행위를 하여서는 안됩니다.

Trademarks

BX Framework™ is a registered trademark of BankwareGlobal Co., Ltd. Other products, titles or services may be registered trademarks of their respective companies.

BX Framework™ 은 BankwareGlobal Co., Ltd.의 등록상표입니다. 기타 모든 제품들과 회사 이름은 각각 해당 소유주의 상표로서 참조용으로만 사용됩니다.

차례

안내서에 대하여	vi
1 코딩표준	1
1.1 BXCM 표준	1
1.2 Java 표준	7
1.3 주석문	10
2 명명규칙	13
2.1 어플리케이션 분류	13
2.2 표준화 대상	13
2.3 공통 명명규칙	15
2.4 온라인 어플리케이션 명명규칙	18
2.5 배치 어플리케이션 명명규칙	34
2.6 기타 명명 규칙	38

그림 목록

[그림 1.1]	@BXMType - LogicalArea	11
[그림 1.2]	@BXMType - IF	12
[그림 1.3]	@BXMType - Loop	12
[그림 2.1]	테스트 케이스 생성 모습	33
[그림 2.2]	배치작업 처리 모델	35

표 목록

[표 1.1]	어노테이션 요약	7
[표 2.1]	어플리케이션 코드	13
[표 2.2]	온라인 어플리케이션 용어	14
[표 2.3]	배치 어플리케이션 용어	15
[표 2.4]	기타 용어	15
[표 2.5]	표현식	16
[표 2.6]	Component 메소드 prefix	17
[표 2.7]	Bean 메소드 prefix	17
[표 2.8]	어플리케이션 명명	18
[표 2.9]	Component IO 명명 규칙	26
[표 2.10]	Service IO 명명 규칙	28
[표 2.11]	DBIO IO 명명 규칙	29
[표 2.12]	인터페이스 IO 명명 규칙	30
[표 2.13]	Component IO 명명 규칙	31
[표 2.14]	테스트 케이스 규칙	33
[표 2.15]	배치 어플리케이션 명명 규칙	35
[표 2.16]	Job 명명 규칙	36
[표 2.17]	Job Configuration File 규칙	37
[표 2.18]	온라인 거래코드	38
[표 2.19]	온라인 로깅	38
[표 2.20]	배치 작업 ID	39
[표 2.21]	배치 로깅	39

안내서에 대하여

안내서의 대상

본 안내서는 아래와 같은 관련자를 대상으로 한다.

- 프레임워크 개발자

안내서 구성

본 안내서는 총 2개의 장으로 구성되어 있다.

- 제1장 : 코딩표준

BXCM, JAVA, 주석문에 대한 개발 코딩 표준에 대하여 설명한다.

- 제2장 : 명명규칙

개발 명명 규칙에 대하여 설명한다.

안내서 규약

표기	의미
<AaBbCc123>	프로그램 소스 코드의 파일명, 디렉터리
<Ctrl>+C	Ctrl과 C를 동시에 누름
[Button]	GUI의 버튼 또는 메뉴 이름
진하게	강조
" "(따옴표)	다른 관련 안내서 또는 안내서 내의 다른 장 및 절 언급
'입력항목'	화면 UI에서 입력 항목에 대한 설명
하이퍼링크	메일계정, 웹 사이트
>	메뉴의 진행 순서
<div>참고</div> <div>참고사항</div>	참고사항
<div>경고</div> <div>경고사항</div>	경고사항
[그림 1.1]	그림 이름
[표 1.1]	표 이름
AaBbCc123	명령어, 명령어 수행 후 화면에 출력된 결과물, 예제코드
<<사용자값>>	예제코드나 설정내용 중 실제 사용자 환경에 맞게 수정되어야 하는 값

표기	의미
[]	옵션 인수 값
	선택 인수 값

연락처

Korea

뱅크웨어글로벌(주)
 100-783 서울특별시 중구 통일로 86, 3층 (순화동 바비엡빌딩 3차)
 Tel: 02-501-6415
 Fax: 02-501-6416
 Email: admin@bankwareglobal.com
 Home : <http://bankwareglobal.com>

1. 코딩표준

1.1 BXCM 표준

1.1.1 Controller, Service, Component , DBIO 사용 공통

필수 작성 사항

Controller, Service, Component , DBIO 사용 시 공통적인 필수 작성 사항에 대해 설명한다.

(1) @BxmCategory

Controller, Service, Component 의 클래스와 각 클래스의 메소드는 @BxmCategory 어노테이션을 포함 해야 한다 (DBIO는 소스생성 방식이므로 개발자가 직접 작성하지 않음).

- Class에 정의된 @BxmCategory 어노테이션

```
@Service
@BxmCategory(logicalName="직원조회 서비스", description="직원조회를 위한 서비스이다.")
public class EmployeeService {
    private Logger logger= LoggerFactory.getLogger(getClass());
    ... ..
}
```

- 메소드에 정의된 @BxmCategory 어노테이션

```
@BxmCategory(logicalName = "직원정보 단건 조회")
public EmpIO retrieveEmp(EmpIO input)
{
    logger.debug("===== SERVICE START =====");
    logger.debug("input = {}", input);
    ... ..
}
```

(2) Application Exception

Service와 Component 등 업무 소스에서는 예외처리를 하지 않는 것이 원칙이나, 예외처리시에는 반드시 bxmc.core.ext.exception.DefaultApplicationException를 throws 해야 한다.

```
@BxmCategory(logicalName = "직원정보 단건 조회")
public EmpIO retrieveEmp(EmpIO input) throws DefaultApplicationException /* Exception */ {
    logger.debug("===== SERVICE START =====");
    logger.debug("input = {}", input);
    ... ..

    try {
        /* 업무 처리 */
    } catch (DefaultApplicationException e) {
```



```
// 메시지 내용
throw new DefaultApplicationException("bxmc.test...", null, e);
}
}
```

금지 항목

Service, Component, DBIO 사용 시 사용하지 말아야 하는 항목에 대해 설명한다.

(1) 객체 직접 생성 금지

Service, Component, DBIO는 프레임워크에서 관리하는 객체이므로 개발자가 new를 통해 직접 객체를 생성하지 않는다.

```
/* 직접 New 연산자를 통해서 생성하면 안된다 */
private EmployeeDbio employeeDbio = new EmployeeDbio();
```

업무 클래스의 객체의 생성은 직접 생성하지 않고 멤버변수 선언부에 getBean을 사용하거나, @Autowired Annotation을 사용하여 가져온다.

- getBean 사용

```
employeeDbio = DefaultApplicationContext.getBean(employeeDbio, EmployeeDbio.class);
```

- Autowired Annotation 사용

```
/* @Autowired Annotation 생성 */
@Autowired
private EmployeeDbio employeeDbio;
```

참고

모놀로식 어플리케이션과 같은 구조라면 getBean() 방식, 마이크로서비스 구조라면 @Autowired Annotation 방식을 권장한다.

(2) 데이터 객체를 멤버변수로 포함 금지

Service, Component, DBIO는 싱글톤 객체로 해당 클래스의 인스턴스는 프레임워크 내에 1개만 생성 된다. 따라서, 데이터를 저장하는 일반객체(원시타입 포함) 및 IO 객체를 멤버 변수로 가지고 있게 되면 객체의 데이터가 공유되는 형태가 되므로 사용하지 않아야 한다(버그발생가능)

(3) 자신의 타입을 멤버 변수에 포함 금지

Service, Component 등의 Bean 클래스의 경우 자신을 멤버 변수로 갖는 구조는 허용되지 않는다.

```
@BxmCategory(logicalName="샘플 Component", description="")
public class EmployeeComponent {

    private final Logger logger = LoggerFactory.getLogger(this.getClass());

    /* EmployeeComponent Bean이 자기 자신(EmployeeComponent)을 멤버변수로 가지고 있으면 안된다 */
    private EmployeeComponent employeeComponent;

    @BxmCategory(logicalName="사원 조회", description="")
    public EmployeeIO getEmployee(int empNo){
        ... ...
    }
}
```

(4) 멤버변수의 get/set 메소드 작성 금지

```
@Service
@BxmCategory(logicalName="직원조회 서비스", description="직원조회를 위한 서비스이다.")
public class EmployeeService {
    private Logger logger= LoggerFactory.getLogger(getClass());

    @Autowired
    private EmployeeComponent employeeComponent;

    /* 멤버변수의 get 메소드가 있으면 안된다 */
    public EmployeeComponent getEmployeeComponent() {
        return employeeComponent;
    }

    /* 멤버변수의 set 메소드가 있으면 안된다 */
    public void setEmployeeComponent(EmployeeComponent employeeComponent) {
        this.employeeComponent = employeeComponent;
    }
    ... ...
}
```

1.1.2 Controller

필수 작성 사항

Controller 작성 시 필수로 작성해야 하는 항목에 대해 설명한다.

- (1) Controller는 @RestController 어노테이션을 클래스 정의부에 포함한다.
- (2) @RequestMapping 어노테이션 내에 Controller의 URL 경로 매핑 정보를 나타내어야 한다.
- (3) Controller의 이름과 클래스 이름은 명명규칙 표준을 따른다.

메소드에 HTTP 요청 메소드처리와 url정보를 설정한다.

금지 항목

Controller 작성 시 금지 항목에 대해 설명한다.

- (1) 멤버변수로 Service의 객체만 가질 수 있다.
- (2) 비즈니스 로직을 작성하지 않는다.

1.1.3 Service

필수 작성 사항

Service 작성 시 필수로 작성해야 하는 항목에 대해 설명한다.

- (1) Service는 @Service 어노테이션을 클래스 정의부에 포함한다.
- (2) Service 이름과 클래스 이름은 명명규칙 표준을 따른다.

금지 항목

Service 작성 시 금지 항목에 대해 설명한다.

- (1) 멤버변수로 Service, Component의 객체만 가질 수 있다. (Controller 호출 금지)
- (2) Service 클래스 메소드의 시작은 표준으로 정의된 동사로 시작해야 한다.

1.1.4 Component

필수 작성 사항

Component 작성 시 필수로 작성해야 하는 항목에 대해 설명한다.

- (1) @Component

Component는 @Component 어노테이션을 클래스 정의부에 포함한다.

```
/* Component 작성 시 필수 어노테이션 */
@Component
@BxmCategory(logicalName = "샘플직원정보관리")
public class EmpComponent {
    private final Logger logger = LoggerFactory.getLogger(this.getClass());
    ... ..
}
```

배치 Component(ItemReader, ItemProcessor, ItemWriter, Tasklet Interface로 구현)일 경우에는 Component 명을 명시한다.

```

/* Component 작성 시 필수 어노테이션 */
@Scope("step")
@Component("SampleComponent")
public class SampleComponent implements ItemProcessor<Sampleinputdto, Sampleoutputdto>, ItemStream,
    ItemReader<Sampleinputdto>, ItemWriter<Sampleoutputdto>
{
    final Logger logger = LoggerFactory.getLogger(this.getClass());

    private Employee001 employee001;
    ... ..
}

```

금지 항목

Component 작성시 금지 항목에 대해 설명한다.

(1) Controller, Service 직접 호출 금지

Component 에서 Service를 호출 할 수 없다.

```

@Component
@BxmCategory(logicalName = "샘플직원정보관리")
public class EmpComponent {
    private final Logger logger = LoggerFactory.getLogger(this.getClass());

    /* Service */
    @Autowired
    private EmployeeService employeeService; // 금지

    @BxmCategory(logicalName = "단건 조회")
    public EmpOut getEmpInf(EmpIn input) throws tApplicationException {

        /* Component에서 Service를 호출하면 안된다 */
        employeeService.getmSmpEmpInfMng();
        return new EmpOut();
    }
}

```

1.1.5 DBIO

금지 항목

다음은 DBIO를 사용시 주의 사항에 대한 설명이다.

(1) DBIO 인터페이스 구현 금지

프레임워크에서 DI(Dependency Injection)을 통해 구현클래스를 주입하므로 DBIO 인터페이스를 구현하는 구현클래스를 개발자가 직접 작성하지 않는다.

(2) 인터페이스 파일 직접 편집 금지

DBIO는 .dbio 파일과 .java(interface로 정의됨)이 한 쌍으로 생성되며 .java 파일은 개발자가 직접 작성/편집하지 않는다.

(3) DBIO 에 생성하는 SQL 의 개수는 10개 미만으로 권장한다.

SQL의 개수가 너무 많으면 DBIO 에디터 동작 및 커밋 동작이 느려질 수 있다.

1.1.6 IO

금지 항목

다음은 IO 사용 시 주의 사항에 대한 설명이다.

(1) 자동 생성된 파일 직접 편집 금지

자동 생성된 .java 파일은 개발자가 직접 수정하지 않는다.

1.1.7 @BxmCategory

@BxmCategory 어노테이션은 플로우다이어그램 생성 시 반드시 필요한 어노테이션이다. 다음은 Controller, Service, Component, 각 메소드, DBIO, DBIO 메소드에 포함되는 @BxmCategory 어노테이션의 사용상 주의 사항을 설명한다.

필수 작성 사항

(1) @BxmCategory 는 logicalName정보를 포함한다.

(2) 논리명인 logicalName은 Package 익스플로러에 노출되는 이름으로 알아보기 쉬운 이름을 사용한다

금지 항목

(1) logicalName은 문자열로 작성하며 Wn, Wt 등의 특수 문자(Escape)를 사용하지 않는다. 또한 '+' 로 문자열을 표현 하지 않는다

```
@Component
@BxmCategory(logicalName = "샘플직원\n정보\n관리")
public class EmpInfMngComponent {
    private final Logger logger = LoggerFactory.getLogger(this.getClass());
    ...
}
```

1.1.8 어노테이션 요약

어노테이션은 JAVA 메타 언어로 소스 상에 표기하는 것만으로 런타임에 어떤 기능을 수행하거나, 클래스로부터 어떤 정보를 추출 할 수 있도록 한다.

[표 1.1] 어노테이션 요약

구분	설명	표기위치
@RestController	Controller클래스로 기능함	Controller 클래스 선언부
@Controller		
@RequestMapping	Controller의 URL 경로 매핑 정보	Controller 클래스 / public 메소드 선언부
@GetMapping	HTTP GET 요청 매핑 정보	Controller public 메소드 선언부
@PostMapping	HTTP GET 요청 매핑 정보	Controller public 메소드 선언부
@Service	Service 클래스로 기능함	Service 클래스 선언부
@Component	Component 클래스로 기능함	Component 클래스 선언부
@BxmDataAccess	DBIO로 기능함	DBIO 인터페이스 선언부 (자동 생성)
@BxmCategory	논리명 입력	Controller 클래스 선언부 Controller 클래스 public 메소드 선언부 Service 클래스 선언부 Service 클래스 public 메소드 선언부 Component 클래스 선언부 Component 클래스 public 메소드 선언부 DBIO 인터페이스 선언부 (자동 생성) DBIO 인터페이스 메소드 선언부 (자동 생성)
@Autowired	Bean 자동 주입.	Controller, Service, Component 의 멤버변수 선언 위치
@Transactional	Transaction을 처리할 때 사용 (Transaction 분리 시 사용)	Transaction을 처리할 메소드 선언부

1.2 Java 표준

1.2.1 금지항목

(1) 로깅을 위해 System.out.println, System.err.println 등을 사용하지 않는다.

```
/* Logger를 사용해서 로그를 해야 한다 */
System.out.println("#Customer ID:" + input.getId());
System.err.println("#Array Count:" + input.getCount())'
```

(2) DB Connection 을 직접 생성하지 않는다.

```
Connection conn = null;
try {
    /* 소스코드 내에서 직접 Connection을 생성하지 않는다 */
    conn = DriverManager.getConnection("jdbc:oracle:thin@127.0.0.1:orcl");
} catch (Exception e) {
    e.printStackTrace();
}
```

(3) 사용하지 않는 멤버 변수나 지역 변수, 메소드는 제거한다.

1.2.2 Immutable 객체 사용시 유의 사항

Immutable 객체란 생성 후 값이 변하지 않는 객체를 말한다. 따라서 Immutable 객체에는 set 메소드가 없으며 멤버 변수를 변경 할 수 없다.

java.math.BigDecimal

금액, 이율 계산 등 정밀한 값의 계산이 필요한 항목은 int, long, float, double(Wrapper Class포함) 등의 원시 타입을 사용하지 않고 java.math.BigDecimal 클래스를 사용한다. 사용 시 주의사항은 다음과 같다.

(1) 연산에 관련된 add, subtract, multiply, divide, round, abs 등의 메소드는 반드시 좌변(Left-hand-side)에 값을 할당 해야 한다.

```
BigDecimal bdOne = new BigDecimal(10);
BigDecimal bdTwo = new BigDecimal(20);

/* 잘못된 예 */
bdOne.add(bdTwo);

/* 올바른 예 */
bdOne = bdOne.add(bdTwo);
```

(2) 값 비교는 객체끼리 '==' 연산 등을 사용하지 않고 compareTo 메소드를 사용한다.

```
BigDecimal bdOne = new BigDecimal(10);
BigDecimal bdTwo = new BigDecimal(20);

/* 잘못된 예 */
if(bdOne == bdTwo) {
    ...
}

/* 올바른 예 */
int result = bdOne.compareTo(bdTwo);
if(result == 0) {
    ...
}
```

(3) BigDecimal 클래스의 자세한 사용법은 Java API를 확인한다.

java.lang.String

BigDecimal 객체와 마찬가지로 String 객체도 Immutable 한 객체이다.

(1) 연산에 관련된 concat, replace 등의 메소드를 사용할 경우에는 반드시 좌변에 값을 할당 해야 한다.

```
String strOne = "ABC";

/* 잘못된 예 */
strOne.replace("A", "Z");

/* 올바른 예 */
strOne = strOne.replace("A", "Z");
```

(2) String 값의 비교는 '==' 연산을 사용하지 않고 equals 메소드를 사용한다.

```
/* 잘못된 예 */
if("CODE00" == code) {
    ...
}
/* 올바른 예 */
if("CODE00".equals(code)) {
    ...
}
```

(3) 문자열 + 문자열 연산은 StringBuilder 클래스를 사용한다.

```
String[] userNameArr = new String[]{"Peter", "Steve", "Sally"};

/* 잘못된 예 */
String msg1 = null;
for(String name : userNameArr) {
    msg1 += name + ",";
}

/* 올바른 예 */
StringBuilder msg2 = new StringBuilder();
for(String name : userNameArr) {
    msg2.append(name).append(",");
}
```

(4) String 클래스의 자세한 사용법은 Java API를 확인한다.

1.3 주석문

이번 장에서는 어플리케이션 개발을 위해 작성되는 프로그램 주석 작성에 대한 주의 사항을 설명한다. 주석문은 JavaDoc 형태의 주석문을 사용한다.

1.3.1 Service, Component 클래스 주석문

Service 와 Component 클래스의 주석문은 Studio 의 Wizard로 생성하면 다음처럼 자동으로 생성 된다. @author 부분에 개발자 id가 들어간다.

```
/**
 * 직원정보를 조회하는 샘플 컴포넌트이다.
 *
 * @author 개발자(id)
 */
```

1.3.2 Service 및 Component의 메소드 주석문

Service와 Component의 메소드의 주석문은 사용자가 직접 입력한다. 주석문을 작성하려는 Service 또는 Component 의 메소드 선언 위에 /**을 입력 후 Enter 키를 입력하면 다음과 같은 JavaDoc 주석이 생성된다. (생성되지 않는 부분은 직접 입력한다.)

```
/**
 * 사번으로 사원 정보를 조회한다.
 *
 * @param empNo
 * @return EmployeeIO
 * @throws ApplicationException
 */
@XmlCategory(logicalName="사원 조회", description="사번으로 사원 정보를 조회한다. ")
public EmployeeIO getEmployee(int empNo) throws ApplicationException {
    ...
}
```

1.3.3 메소드 본문에서의 주석문

Service 및 Component 의 메소드 본문에서 주석은 Flow Diagram 산출물을 위해 다음과 같이 JavaDoc 형태의 주석을 사용한다.

1.3.4 @BXMType

소스코드에 주석 및 어노테이션을 작성하면 이를 이용하며 Studio에서 소스코드의 플로우를 표현한다.

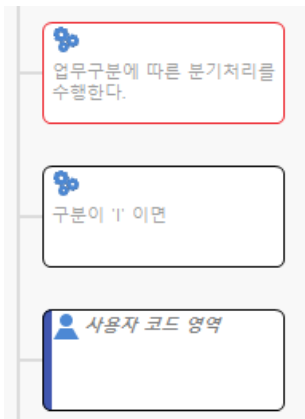
LogicalArea

프로우 디자이너로 업무 흐름을 표현하고자 할 때 사용한다.

@Desc : BXMType의 설명을 작성한다. 한줄 인 경우에 20자 이내에서 작성하고, 두줄인 경우 한줄에 10자 이내에서 작성한다. . 초과한 경우 플로우 디자이너에서 잘려서 보인다.

```
/**
 * @BXMType LogicalArea
 * @Desc 업무구분에 따른 분기처리를 수행한다.
 */

/**
 * @BXMType LogicalArea
 * @Desc 구분이 'I' 이면
 *   등록처리
 */
logger.debug("등록처리");
```



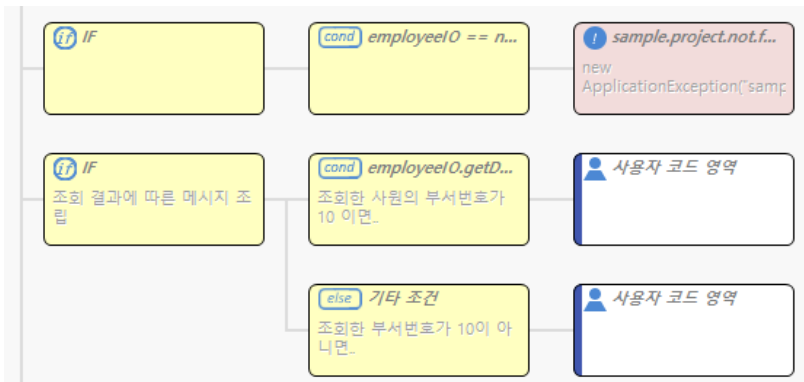
[그림 1.1] @BXMType - LogicalArea

IF

플로우 디자이너에 표현하고자 하는 분기문에 대해서 아래와 같이 작성한다.

@Desc : BXMType의 설명을 작성한다. 한줄 인 경우에 20자 이내에서 작성하고, 두줄인 경우 한줄에 10자 이내에서 작성한다. 초과한 경우 플로우 디자이너에서 잘려서 보인다.

```
/**
 * @BXMType IF
 * @Desc 조회 결과에 따른 메시지 조립
 */
if(employeeIO.getDeptNo() == 10)/** 조회한 사원의 부서번호가 10 이면.. */
{
    logger.debug("조회한 사원의 부서번호 : 10");
}
else /** 조회한 부서번호가 10이 아니면.. */
{
    logger.debug("조회한 사원의 부서번호 : {}", employeeIO.getDeptNo());
}
```



[그림 1.2] @BXMTType - IF

Loop

플로우 디자이너에 표현하고자 하는 반복문에 대해서 아래와 같이 작성한다.

@Desc : BXMTType의 설명을 작성한다. 한줄 인 경우에 20자 이내에서 작성하고, 두줄인 경우 한줄에 10자 이내에서 작성한다. 초과한 경우 플로우 디자이너에서 잘려서 보인다.

```
/**
 * @BXMTType Loop
 * @Desc Component의 Output을 Service의 Output으로 설정.
 */
for (EmployeeIO fetchDto : compoOutput) {
    outDtoSub1 = new EmployeeIO();

    outDtoSub1.setEmpNo(fetchDto.getEmpNo());
    outDtoSub1.setEmpNm(fetchDto.getEmpNm());
    outDtoSub1.setOcpNm(fetchDto.getOcpNm());

    output.getGrid01().add(outDtoSub1);
}
```



[그림 1.3] @BXMTType - Loop

2. 명명규칙

2.1 어플리케이션 분류

파일명이나 Package, 클래스 이름 등에 업무와 연관 지어 명명을 할 경우에는 다음 표에 정의된 어플리케이션 구분 코드를 사용하여 작성한다.

다음 코드는 BXCM 표준 샘플 프로젝트의 코드이다.

[표 2.1] 어플리케이션 코드

L1코드	L1한글명	L1영문명	L2코드	L2한글명	L2영문명
bxcn	프레임워크	Business Excellence Cloud Framework	smp	샘플	Sample

참고

Kubernetes 환경에서는 대문자/특수문자를 지원하지 않는다. 프로젝트명이나 Job명은 등은 소문자와 '-' 의 조합으로 구성하는 것을 권장한다.

2.2 표준화 대상

어플리케이션 작성을 위한 명명규칙에 대한 설명을 한다. 명명 규칙은 다음의 내용을 포함한다.

2.2.1 온라인 어플리케이션

[표 2.2] 온라인 어플리케이션 용어

구분	설명
어플리케이션	<ul style="list-style-type: none"> • 사용자의 요청을 처리하기 위한 프로그램의 집합으로 데이터 액세스와 처리를 위해 작성된 SQL, 비즈니스 로직 구현을 위한 클래스, 설정파일들을 포함하여 구성된다. • 어플리케이션은 Eclipse의 BX 클라우드 프로젝트로 작성된다. • 본 프로젝트의 업무 개발자는 별도로 어플리케이션을 생성하지 않고 이미 작성되어 있는 Git Repositor에 제공된 어플리케이션을 Clone하여 사용한다. • 외부에서 식별하기 위해 시스템 내에서 유일한 이름을 가진다.
컨트롤러 (Controller)	<ul style="list-style-type: none"> • Rest Client(사용자 화면, 대내외채널 등 외부에서 호출된 사용자 요청)의 요청에 대응되는 클래스이다. • HTTP 클라이언트 요청을 처리(HTTP 의존적인 영역을 처리)한다. • 트랜잭션 처리 단위의 기준이 된다. • 비즈니스 로직을 처리하기 위해 Service를 호출한다. • Component와 DBIO는 직접 처리하지 않는다.
서비스(Service)	<ul style="list-style-type: none"> • Controller와 1:1 로 매핑되는 클래스이다 • Controller에서 호출되는 단위의 기능을 Service의 메소드로 구현한다.
컴포넌트 (Component)	<ul style="list-style-type: none"> • 업무 기능 중 공통적으로 사용 할 수 있는 기능을 추상화하여 제공하는 클래스이다. • 재사용이 가능한 업무 단위라고 볼 수 있다.
DBIO	<ul style="list-style-type: none"> • 데이터 액세스를 위해 작성하는 개발 리소스로 자바 Interface와 SQL Mapper(.dbio)파일로 구성된다. • 개발도구에서 DBIO를 생성하여 개발한다. • SQL은 DBIO에서 SQL ID를 추가하여 작성한다.
SQL ID	<ul style="list-style-type: none"> • DBIO에서 생성되는 쿼리를 나타내는 식별자이다. DBIO 인터페이스의 메소드 이름이 된다.
IO	<ul style="list-style-type: none"> • 메시지(JSON, 고정길이 데이터, XML 등) 형태로 변환할 수 있는 기능을 제공하는 프레임워크 표준 데이터 전달 객체로 Controller 요청/응답, 대내외계 인터페이스, 비즈니스 호출 등에 사용된다. • 모든 DTO(Data Transfer Object)는 IO로 관리된다.
일반 클래스	<ul style="list-style-type: none"> • 순수 자바 클래스로 사용자가 자유롭게 작성이 가능하다.
JUnit 테스트 케이스	JUnit을 사용해 업무 소스를 테스트 할 수 있는 테스트 파일.

2.2.2 배치 어플리케이션

[표 2.3] 배치 어플리케이션 용어

구분	설명
어플리케이션	배치 작업을 위한 프로그램의 집합으로 배치작업 설정파일, 데이터 액세스와 처리를 위해 작성된 SQL, 비즈니스 로직 구현을 위한 클래스들을 포함하여 구성된다.
Job	배치 업무를 처리하기 위한 플로우로 구성된 배치작업의 실행 및 재실행 단위로 플로우를 기술하는 XML 설정으로 작성된다.
Step	Job을 구성하는 업무 구현의 최소단위이다.
Job Configuration 설정 파일	Job 과 Step 의 옵션 및 실행 순서를 설정하는 파일이다.

2.2.3 기타

[표 2.4] 기타 용어

구분	설명
거래ID	특정 거래를 처리하는 Rest API 식별자
작업 ID	배치 Job을 구분하는 식별자
로그 파일	거래 로그 혹은 배치 수행 로그를 저장하는 파일

2.3 공통 명명규칙

모든 명명규칙은 다음의 공통 사항을 포함한다.

2.3.1 제약사항

- (1) 메타에서 제공하는 메타 데이터를 제외한 나머지 항목은 한글 발음으로 된 영문자 사용을 제한한다. 예)직업- >Jikup
- (2) 영문자를 제외한 기호, 특수 문자 등은 사용을 제한한다. 예) _, \$
- (3) 테이블이름을 사용하는 경우 예외로 한다
- (4) 표준단어 영문약어 조합으로 각 컴포넌트를 명명한다.

2.3.2 표준단어

- (1) 표준단어를 다음의 카멜 표기법으로 변환하여 사용한다.
- (2) 카멜 표기법 변환 규칙: _(언더바)로 구분된 단어에서..

- A. 첫 번째 단어는 모두 소문자 표기.
- B. 두 번째 단어부터 첫 글자는 대문자, 나머지는 소문자 표기.
- C. 외자인 단어는 대문자로 표기.

(3) 카멜 표기법에 의한 표준단어 변환 예시

- A. USR_NM → usrNm
- B. DPST_AMT → dpstAmt
- C. N_CASTING → nCasting
- D. CUST_N_ID → custNId

2.3.3 표현식

명명규칙에 사용되는 표현식은 다음과 같다.

[표 2.5] 표현식

표현식	설명	구성	예시
B	B가 반드시 있음.	A + B + C	ABC
(B)*	B가 0개 이상 존재.	A + (B)* + C	AC, ABC, ABB...C
(B)?	B가 1개 있거나 없음.	A + (B)? + C	AC, AB
(BC)*	BC 가 0개 이상 존재.	A + (BC)* + C	AC, ABCC, ABCBC...C
(B C)	B 또는 C.	A + (B C) + C	ABC, ACC
(B C)*	B 또는 C 가 0개 이상 존재.	A + (B C)* + C	AC, ABC, ABCBC 등
[0-9]	0부터 9까지의 숫자	A + [0-9] + C	A0C, A1C, ..., A9C,
[00-99]	00 부터 99까지의 숫자	A + [00-99] + C	A00C, ..., A09C, ..., A99C
[A-Z]	알파벳A부터 Z까지	A + [A-Z] + C	AAC, ..., AJC, ..., AZC
[a-zA-Z]	알파벳 a-z 또는 A-Z까지	A + [a-zA-Z] + C	AaC, AAC, ..., AjC, ..., AJC, ..., AzC, ..., AZC

2.3.4 Prefix

메소드의 역할 별로 prefix로 사용되는 동사가 결정된다.

Controller, Service, Component 메소드 prefix

Controller, Service, Component 메소드는 메소드 역할에 따라 아래의 Prefix를 사용한다.

[표 2.6] Component 메소드 prefix

메소드역할	prefix	예시
조회	get	getUser, getUserList, getUsers
설정	set	setUser, setUserList, setUsers
추가	add	addUser
삭제	remove	removeUser
생성/신규	create	createUser
수정	modify	modifyUserInfo
초기화	init	initProcessor
인증/검증	validate	validateUserInfo
검사	check	checkId
처리	process	processCalcuation
계산	calc	calcInterest
포함여부	has	hasUser, hasElement, hasId
참/거짓	is	isTrue, isFalse
존재여부	exist	existUser, existed
호출	call	callEAI0001, callRULE0001

DBIO 메소드 prefix

DBIO 메소드의 Prefix 는 statement 타입에 따라 아래의 Prefix 를 사용한다.

[표 2.7] Bean 메소드 prefix

역할	prefix	예시
단건조회	selectOne	selectOne01
다건조회	selectList	selectList01
페이징조회	selectPage	selectPage01
단건조회 Lock Update	selectOneLock	selectOneLock01
다건조회 Lock Update	selectListLock	selectListLock01
추가	insert	insert01
수정	update	update01
삭제	delete	delete01

기타 메소드 prefix

일반 클래스의 메소드는 Prefix 제한을 두지 않는다. 의미 있는 표준단어를 사용한다.

2.4 온라인 어플리케이션 명명규칙

2.4.1 어플리케이션

어플리케이션 단위는 관리 주체가 다르고, 어플리케이션 간 기능 호출을 최소화 할 수 있는 단위로 구분 한다. 어플리케이션은 Studio의 'BX 클라우드 프로젝트' 메뉴를 통해 생성한다.

[표 2.8] 어플리케이션 명명

구분	설명
구성	L1코드 + '-' + L2코드(영문명) + '-' + online
적용사례	bxcn-sample-online
기본원칙	<ol style="list-style-type: none">1. 어플리케이션 이름 작성은 다음과 같은 표기법을 따른다. L1코드,L2코드(영문명)의 소문자로 표기 단어와 단어 사이에 '-' 로 구분2. 어플리케이션 이름은 중복되어서는 안되므로 전체 업무를 통틀어 유일한 이름을 갖도록 작성한다.

2.4.2 Controller

Controller는 Studio 의 'Controller 클래스' 메뉴를 통해 생성 되며 Controller의 클래스 이름과 Controller 이름은 다음과 같이 작성한다.

구분	설명
Package - 구성	L1코드 + . + L2코드 + . + online + . + controller
Package - 적용사례	bxcn.sample.online.controller
Package - 기본원칙	<ul style="list-style-type: none">- Package 이름 작성은 모두 소문자로 한다.- 분류는 "."로 구분한다.- Package의 마지막 세그먼트는 controller를 쓴다.
클래스 - 구성	L2코드(영문명) + (표준단어)* 조합 + Controller
클래스 - 적용사례	SmpEmployeeController, SmpDeptController
클래스 - 기본원칙	<ul style="list-style-type: none">- L2코드(영문명), 표준단어 조합, 클래스 구分的 조합으로 구성한다.- L2코드(영문명)는 첫 글자는 대문자, 나머지는 소문자로 표기한다.- Controller는 @RestController, @Controller 어노테이션을 포함한다.- Controller 이름은 어플리케이션 내에서 중복이 되면 안되므로 유일한 이름을 갖도록 한다.- RequestMapping의 이름은 클래스명의 표준단어 조합의 소문자로 구성한다.

구분	설명
	<ul style="list-style-type: none"> - 파일의 확장자는 java를 사용한다.
클래스 - 부가정보	<ul style="list-style-type: none"> - 부가정보는 @BxmCategory 어노테이션을 사용하여 기입한다. - @BxmCategory 는 필수 항목으로 작성한다. - 부가정보는 다음과 같다. <ol style="list-style-type: none"> 1. 논리명(logicalName) : Controller의 논리명을 입력한다.
클래스 - 작성 예	<pre> @RestController @RequestMapping("/employee") @BxmCategory(logicalName = "직원관리컨트롤러") public class SmpEmployeeController { /* 클래스 이름 */ ... }</pre>
URL - 구성	역할에 따른 HTTP메소드 + (메소드 식별 단어)*
URL - 적용사례	조회 : GetMapping("/get") 등록 : PutMapping("/add") 수정 : PostMapping("/modify") 삭제 : DeleteMapping("/remove")
메소드 - 구성	메소드 prefix + (표준단어)*
메소드 - 적용사례	getEmployee - 샘플 직원정보를 얻는다. addEmployee - 샘플 직원정보를 등록한다.
메소드 - 기본원칙	<ul style="list-style-type: none"> - 메소드 Prefix 는 메소드 Prefix 장을 참고하여 작성한다. 메소드 동작에 해당하는 Prefix를 선택한다. - 표준단어는 자연어의 조합을 사용한다. - 하나의 Controller의 메소드에는 너무 많은 코드를 작성하지 않는다.
메소드 - 부가정보	<ul style="list-style-type: none"> - 부가정보는 @BxmCategory 어노테이션을 사용하여 기입한다. - @BxmCategory 는 필수 항목으로 작성한다. - 부가정보는 다음과 같다. <ol style="list-style-type: none"> 1. 논리명(logicalName): Service 메소드의 논리명을 입력한다.
메소드 - 작성 예	<pre> @PostMapping("/add") @BxmCategory(logicalName = "직원정보등록") public SmpEmployeeController01OutDto addEmployee(@RequestBody SmpEmployeeController01InDto input) { ... }</pre> <pre> @GetMapping("/get/{empNo}") @BxmCategory(logicalName = "직원정보등록") public SmpEmployeeController01OutDto getEmployee(@PathVariable int empNo) {</pre>

구분	설명
	<pre> ... } </pre>

2.4.3 Service

Service는 Studio 의 'Service 클래스' 메뉴를 통해 생성 되며 Service의 클래스 이름과 Service 이름은 다음과 같이 작성한다.

구분	설명
Package - 구성	L1코드 + . + L2코드 + . + online 또는 batch + . + service
Package - 적용사례	bxcn.sample.online.service
Package - 기본원칙	<ul style="list-style-type: none"> - Package 이름 작성은 모두 소문자로 한다. - 분류는 "."로 구분한다. - Package의 마지막 세그먼트는 service를 쓴다.
클래스 - 구성	L2코드(영문명) + (표준단어)* 조합 + Service
클래스 - 적용사례	SmpEmployeeService, SmpDeptService
클래스 - 기본원칙	<ul style="list-style-type: none"> - L2코드(영문명), 표준단어 조합, 클래스 구분의 조합으로 구성한다. - Service의 메소드가 많아 분리가 필요한 경우 일련번호 2자리를 지정한다. - L2코드(영문명)는 첫 글자는 대문자, 나머지는 소문자로 표기한다. - Service는 @Service 어노테이션을 포함한다. - 파일의 확장자는 java를 사용한다.
클래스 - 부가정보	<ul style="list-style-type: none"> - 부가정보는 @BxmCategory 어노테이션을 사용하여 기입한다. - @BxmCategory 는 필수 항목으로 작성한다. - 부가정보는 다음과 같다. <ul style="list-style-type: none"> 1. 논리명(logicalName) : Service의 논리명을 입력한다.
클래스 - 작성 예	<pre> @Service @BxmCategory(logicalName = "고객관리서비스") public class SmpEmployeeService { /* 클래스 이름 */ ... } </pre>
메소드 - 구성	메소드 prefix + (표준단어)*
메소드 - 적용사례	<p>getEmployee - 샘플 직원정보를 얻는다.</p> <p>addEmployee - 샘플 직원정보를 추가한다.</p>

구분	설명
메소드 - 기본원칙	<ul style="list-style-type: none"> - 메소드 Prefix 는 메소드 Prefix 장을 참고하여 작성한다. 메소드 동작에 해당하는 Prefix를 선택한다. - 표준단어는 자연어의 조합을 사용한다. - 하나의 Service에 Service 메소드는 10개 이하로 작성한다. - 하나의 Service 메소드에 너무 많은 코드를 작성하지 않는다.
메소드 - 부가정보	<ul style="list-style-type: none"> - 부가정보는 @BxmCategory 어노테이션을 사용하여 기입한다. - @BxmCategory 는 필수 항목으로 작성한다. - 부가정보는 다음과 같다. <ul style="list-style-type: none"> 1. 논리명(logicalName): Service 메소드의 논리명을 입력한다.
메소드 - 작성 예	<pre>@BxmCategory(logicalName="고객정보조회") public EmployeeIO getEmployee(EmployeeIO input) throws DefaultApplicationException { ... }</pre>

2.4.4 Component

프레임워크에서 BC(비즈니스 컴포넌트)는 Component으로 사용되며 @Component 어노테이션으로 정의 한다. Component는 Studio 의 '컴포넌트' 메뉴를 통해 생성되며 Component의 이름은 다음과 같이 작성한다.

구분	설명
Package - 구성	L1코드 + . + L2코드(영문명) + . + online 또는 batch + . + component
Package - 적용사례	bxcmm.sample.online.component
Package - 기본원칙	<ul style="list-style-type: none"> - Package 이름 작성은 모두 소문자로 한다. - 분류는 "."로 구분한다. - Package의 마지막 세그먼트는 component를 쓴다.
클래스 - 구성	L3코드(영문명) + (표준단어)* 조합 + (일련번호 2자리) + Component
클래스 - 적용사례	SmpEmployeeComponent, SmpDeptComponent
클래스 - 기본원칙	<ul style="list-style-type: none"> - L3코드 3자리, 표준단어 조합, 클래스 구분의 조합으로 구성한다. - Component의 메소드가 많아 분리가 필요한 경우 일련번호 2자리를 지정한다. - L3코드는 첫 글자는 대문자, 나머지는 소문자로 표기한다. - Component은 @Component 어노테이션을 포함한다. - 파일의 확장자는 java를 사용한다.

구분	설명
클래스 - 부가정보	<ul style="list-style-type: none"> - 부가정보는 @BxmCategory 어노테이션을 사용하여 기입한다. - @BxmCategory 는 필수 항목으로 작성한다. - 부가정보는 다음과 같다. <p>1. 논리명(logicalName) : Component의 논리명을 입력한다.</p>
클래스 - 작성 예	<pre> @Component @BxmCategory(logicalName = "샘플직원정보관리") public class SmpEmployeeComponent { ... }</pre>
메소드 - 구성	메소드 prefix + (표준단어)*
메소드 - 적용사례	<p>getEmployee - 샘플 직원정보를 얻는다.</p> <p>addEmployee - 샘플 직원정보를 추가한다.</p>
메소드 - 기본원칙	<ul style="list-style-type: none"> - 메소드 Prefix 는 메소드 Prefix 장을 참고하여 작성한다. 메소드 동작에 해당하는 Prefix를 선택한다. - 표준단어는 자연어의 조합을 사용한다. - 하나의 Component에 Component 메소드는 10개 이하로 작성한다. - 하나의 Component 메소드에 너무 많은 코드를 작성하지 않는다.
메소드 - 부가정보	<ul style="list-style-type: none"> - 부가정보는 @BxmCategory 어노테이션을 사용하여 기입한다. - @BxmCategory 는 필수 항목으로 작성한다. - 부가정보는 다음과 같다. <p>1. 논리명(logicalName): 메소드의 논리명을 입력한다.</p>
메소드 - 작성 예	<pre> @BxmCategory(logicalName = "단건 등록") public int addEmpEmployee(DSmpEmpTst000Dto input) throws DefaultApplicationException { ... }</pre>

2.4.5 DBIO

DBIO는 데이터베이스에 직접 접근하는 객체로 Bean을 통해서만 호출이 가능하다. DBIO는 Studio 의 'DBIO' 메뉴를 통해 생성되며 DBIO의 이름은 다음과 같이 작성한다.

구분	설명
Package - 구성	L1코드 + . + L2코드(영문명) + . + online 또는 batch + . + dbio

구분	설명
Package - 적용사례	bxcn.sample.online.dbio
Package - 기본원칙	<ul style="list-style-type: none"> - Package 이름 작성은 모두 소문자로 한다. - 분류는 "."로 구분한다. - Package의 마지막 세그먼트는 dbio를 쓴다.
클래스 - 구성	<ol style="list-style-type: none"> 1. PK CRUD : D + 테이블 이름 (본 가이드에선 SMP_EMP_TST 테이블 사용) + 000 2. 그외 : D + 테이블 이름 + (001-999)
클래스 - 적용사례	<ol style="list-style-type: none"> 1. PK CRUD : DSmpEmpTst000 2. 그외 : DSmpEmpTst002
클래스 - 기본원칙	<ul style="list-style-type: none"> - DBIO 이름은 사용하는 쿼리에서 DBIO를 뜻하는 기호 'D' 1자리, 주로 사용하는 테이블의 테이블명과 일련번호 3자리로 구성된다. - 테이블 이름은 카멜 표기법으로 한다. - 일련번호000은 PK로 CRUD를 하는 DBIO를 위한 번호이므로 개발자가 직접 작성/수정 하지 않는다.(Default DBIO 기능으로 자동 생성). - 개발자는 일련번호 001부터 시작하는 DBIO를 작성한다. - 파일의 확장자는 dbio를 사용한다.
클래스 - 부가정보	- DBIO는 소스코드 생성 방식으로 생성 되므로 사용자가 부가 정보를 입력하지 않는다.
클래스 - 작성 예	소스코드를 직접 생성하지 않는다.
SQL ID - 구성	<ol style="list-style-type: none"> 1. DBIO prefix + (일련번호 2자리) 2. DBIO Prefix + (표준단어)* 또는 3. DBIO Prefix + (표준단어)* + (일련번호2자리)
SQL ID - 적용사례	selectOne01 selectOneEmpNm selectOneEmpNm01
SQL ID - 기본원칙	<ul style="list-style-type: none"> - DBIO SQL ID의 이름은 Prefix와 일련번호 2자리로 구성하거나 Prefix와 표준단어 조합, 또는 Prefix와 표준단어 조합에 일련번호 2자리로 구성할 수 있다. - 표준단어는 메타시스템에 정의된 표준단어 영문약어의 조합을 사용한다. - 일련번호 00은 PK를 사용하는 쿼리를 위한 번호이므로 비워둔다. - Prefix는 매핑 되는 SQL의 statement와 일치하게 작성한다. - Prefix는 DBIO 메소드 Prefix 항목을 참고한다.

2.4.6 Default DBIO

테이블을 기반으로 DBIO와 IO를 자동 생성 해주는 Studio의 편의 기능중 하나로 Default DBIO의 표준은 다음과 같다.

(1) 생성된 DBIO

구분	설명
Package - 구성	L1코드 + . + L2코드 + . + online 또는 batch + . + dbio
Package - 적용사례	bxcn.sample.online.dbio
Package - 기본원칙	<ul style="list-style-type: none"> - Package 이름 작성은 모두 소문자로 한다. - 분류는 "."로 구분한다. - Package의 마지막 세그먼트는 dbio를 쓴다.
클래스 - 구성	- D + 테이블 이름 (본 가이드에선 SMP_EMP_TST 테이블 사용) + 000
클래스 - 적용사례	- PK CRUD : DSmpEmpTst000
클래스 - 기본원칙	<ul style="list-style-type: none"> - DBIO 이름은 사용하는 쿼리에서 DBIO를 뜻하는 기호 'D' 1자리, 주로 사용하는 테이블의 테이블명과 일련번호 000으로 구성된다. - 테이블 이름은 카멜 표기법으로 한다. - 파일의 확장자는 dbio를 사용한다.
클래스 - 부가정보	- DBIO는 소스코드 생성 방식으로 생성 되므로 사용자가 부가 정보를 입력하지 않는다.
클래스 - 작성 예	소스코드를 직접 생성하지 않는다.
SQL ID - 구성	<ol style="list-style-type: none"> 1. 메소드 Prefix + _ + PK 이름 2. 메소드 prefix + _ + UK 이름 3. 메소드 Prefix + _ + 인덱스 이름
SQL ID - 적용사례	selectOneLock_SYS_C0011417 insert_SYS_C0011417
SQL ID - 기본원칙	<ul style="list-style-type: none"> - DBIO SQL ID의 이름은 Prefix와 테이블의 PK, UK 사용 여부, PK 또는 UK 이름으로 정해진다. - 표준단어는 메타시스템에 정의된 표준단어 영문약어의 조합을 사용한다. - Prefix는 매핑 되는 SQL의 statement 와 일치하게 작성한다. - Prefix는 DBIO 메소드 Prefix 항목을 참고한다.

(2) 생성된 DBIO의 IO

구분	설명
Package - 구성	L1코드 + . + L2코드(영문명) + . online 또는 batch + . + dbio.dto
Package - 적용사례	bxcmm.sample.online.dbio.dto
Package - 기본원칙	<ul style="list-style-type: none"> - Package 이름 작성은 모두 소문자로 한다. - 분류는 "."로 구분한다. - Package의 마지막 세그먼트는 dbio.dto를 쓴다.
클래스 - 구성	- D + 테이블 이름 (본 가이드에선 SMP_EMP_TST 테이블 사용) + 000 + Dto
클래스 - 적용사례	DSmpEmpTst000Dto
클래스 - 기본원칙	<ul style="list-style-type: none"> - 테이블 이름 (본 가이드에선 SMP_EMP_TST 테이블 사용) + 000에 Dto를 붙인다. - 테이블 이름은 카멜 표기법으로 한다. - 파일의 확장자는 omm를 사용한다.
클래스 - 부가정보	- IO 자바 소스는 소스코드 생성 방식으로 생성 되므로 사용자가 부가 정보를 입력하지 않는다.
클래스 - 작성 예	소스코드를 직접 생성하지 않는다.

2.4.7 IO

IO 이름은 다음의 경우에 따라 구분되어 작성된다. IO는 Controller, Service, Component, DBIO의 IO를 재사용 하도록 표준을 정하여 IO 생성을 최소화 하고, 필요한 경우에만 따로 만들도록 한다.

- (1) Controller Input/Output/공통: Controller의 입출력을 위한 IO
- (2) Service Input/Output/공통: Service의 입출력을 위한 IO
- (3) DBIO Input/Output: DBIO의 입출력을 위한 IO
- (4) 인터페이스 Input/Output: EAI/MCI 등의 인터페이스를 위한 IO
- (5) Component Input/Output/공통: Component의 입출력을 위한 IO

IO는 'IO' 메뉴를 통해 생성되며 IO의 이름은 다음과 같이 작성한다.

Controller를 위한 IO 작성

Controller 메소드의 입력은 기본타입, 일반 객체를 자유롭게 사용할 수 있다.

[표 2.9] Component IO 명명 규칙

구분	설명
Package - 구성	L1코드 + . + L2코드 + . + online 또는 batch + . + controller.dto
Package - 적용사례	bxcn.sample.online.controller.dto
Package - 기본원칙	<ul style="list-style-type: none"> - Package 이름 작성은 모두 소문자로 한다. - 분류는 "."로 구분한다. - Package의 마지막 세그먼트는 controller.dto를 쓴다.
클래스 - 구성	<ul style="list-style-type: none"> - 입력 IO: Controller 이름 + (식별가능한 표준단어)? + 일련번호2자리 + InDto - 출력 IO: Controller 이름 + (식별가능한 표준단어)? + 일련번호2자리 + OutDto - 입력 반복 IO: Controller 이름 + List + 일련번호2자리 + InDto - 출력 반복 IO: Controller 이름 + List + 일련번호2자리 + OutDto - 공통 IO : Controller 이름 + Dto
클래스 - 적용사례	<ul style="list-style-type: none"> - 입력 IO: SmpEmployeeController01InDto - 출력 IO: SmpEmployeeController01OutDto - 입력 반복 IO: SmpEmployeeControllerList01InDto - 출력 반복 IO: SmpEmployeeControllerList01OutDto - 공통 IO: SmpEmployeeControllerDto
클래스 - 기본원칙	<ul style="list-style-type: none"> - Controller의 입출력에 사용되는 IO는 Controller이름, 일련번호, 입출력 Suffix 로 이루어진다. - 입력에 사용되는 IO는 Suffix 를 'InDto' 으로 사용한다. - 출력에 사용되는 IO는 Suffix 를 'OutDto' 으로 사용한다. - 반복 IO는 Controller 이름 뒤에 List로 사용한다. - 작성된 반복 IO는 Controller 입출력 IO에 포함될 수 있다. - 공통 IO는 해당 Controller 전체 도메인에 해당하는 IO로, 해당 Controller 어디에서든 사용 할 수 있다. - 파일의 확장자는 omm 을 사용한다.
클래스 - 부가정보	- IO는 소스코드 생성 방식으로 생성 되므로 사용자가 부가 정보를 입력하지 않는다.
클래스 - 작성 예	소스코드를 직접 생성하지 않는다.
메소드 - 구성	IO는 자바 소스코드가 자동생성 되므로 개발자가 직접 메소드를 구성할 필요가 없다.

Service Input/Output을 위한 IO 작성

Service 메소드의 입력은 기본타입, 일반 객체를 자유롭게 사용할 수 있다. Service 메소드의 IO가 필요한 경우에는 Controller, Component의 IO와 DBIO의 IO를 재사용한다. 재사용이 어려운 경우 다음을 참고하여 생성하여 사용한다.

[표 2.10] Service IO 명명 규칙

구분	설명
Package - 구성	L1코드 + . + L2코드(영문명) + . + online 또는 batch + . + service.dto
Package - 적용사례	bxcn.sample.online.service.dto
Package - 기본원칙	<ul style="list-style-type: none"> - Package 이름 작성은 모두 소문자로 한다. - 분류는 "."로 구분한다. - Package의 마지막 세그먼트는 service.dto를 쓴다.
클래스 - 구성	<ul style="list-style-type: none"> - 입력 IO: Service 이름 + (식별가능한 표준단어) + 일련번호2자리 + InDto - 출력 IO: Service 이름 + (식별가능한 표준단어) + 일련번호2자리 + OutDto - 입력 반복 IO: Service 이름 + List + 일련번호2자리 + InDto - 출력 반복 IO: Service 이름 + List + 일련번호2자리 + OutDto - 공통 IO: Service 이름 + Dto
클래스 - 적용사례	<ul style="list-style-type: none"> - 입력 IO: SmpEmployeeService01InDto - 출력 IO: SmpEmployeeService01OutDto - 입력 반복 IO: SmpEmployeeServiceList01InDto - 출력 반복 IO: SmpEmployeeServiceList01OutDto - 공통 IO: SmpEmployeeServiceDto
클래스 - 기본원칙	<ul style="list-style-type: none"> - Service의 입출력에 사용되는 IO는 Service이름, 일련번호, 입출력 Suffix 로 이루어진다. - 입력에 사용되는 IO는 Suffix 를 'InDto' 으로 사용한다. - 출력에 사용되는 IO는 Suffix 를 'OutDto' 으로 사용한다. - 반복 IO는 Service 이름 뒤에 List로 사용한다. - 작성된 반복 IO는 Service 입출력 IO에 포함될 수 있다. - 공통 IO는 해당 Service 전체 도메인에 해당하는 IO로, 해당 Service 어디에 서든 사용 할 수 있다. - 파일의 확장자는 omm 을 사용한다.
클래스 - 부가정보	- IO는 소스코드 생성 방식으로 생성 되므로 사용자가 부가 정보를 입력하지 않는다.
클래스 - 작성 예	소스코드를 직접 생성하지 않는다.
메소드 - 구성	IO는 자바 소스코드가 자동생성 되므로 개발자가 직접 메소드를 구성할 필요가 없다.

DBIO를 위한 IO 작성

[표 2.11] DBIO IO 명명 규칙

구분	설명
Package - 구성	L1코드 + . + L2코드 + . + online 또는 batch + . + dbio.dto
Package - 적용사례	bxcn.sample.online.dbio.dto
Package - 기본원칙	<ul style="list-style-type: none"> - Package 이름 작성은 모두 소문자로 한다. - 분류는 "."로 구분한다. - Package의 마지막 세그먼트는 dbio.dto를 쓴다.
클래스 - 구성	<ul style="list-style-type: none"> - NAME: DBIO 이름 - PK CRUD: NAME+ Dto - 그 외 <ol style="list-style-type: none"> 1. 입력 IO: NAME + DBIO 메소드 이름 + InDto 2. 출력 IO: NAME + DBIO 메소드 이름 + OutDto
클래스 - 적용사례	<ul style="list-style-type: none"> - PK CRUD : DSmpEmpTst000Dto - 그 외 <ol style="list-style-type: none"> 1. 입력 IO: DSmpEmpTst001selectPage01InDto 2. 출력 IO: DSmpEmpTst001selectPage01OutDto
클래스 - 기본원칙	<ul style="list-style-type: none"> - PK로 CRUD를 하는 일련번호 000의 DBIO IO는 개발자가 직접 작성 하지 않는다. - 개발자가 작성하는 일련번호 001 이후의 DBIO 메소드의 입출력에 사용되는 IO는 DBIO이름, DBIO메소드 이름 및 입출력 Suffix 로 이루어진다. - 입력에 사용되는 IO는 Suffix 를 'InDto' 으로 사용한다. - 출력에 사용되는 IO는 Suffix 를 'OutDto' 으로 사용한다. - 파일의 확장자는 omm 을 사용한다.
클래스 - 부가정보	- IO는 소스코드 생성 방식으로 생성 되므로 사용자가 부가 정보를 입력하지 않는다.
클래스 - 작성 예	소스코드를 직접 생성하지 않는다.
메소드 - 구성	IO는 자바 소스코드가 자동생성 되므로 개발자가 직접 메소드를 구성할 필요가 없다.

인터페이스를 위한 IO 작성

[표 2.12] 인터페이스 IO 명명 규칙

구분	설명
Package - 구성	L1코드 + . + L2코드 + . + online 또는 batch + . + inf.dto
Package - 적용사례	bxcn.sample.online.inf.dto
Package - 기본원칙	<ul style="list-style-type: none"> - Package 이름 작성은 모두 소문자로 한다. - 분류는 "."로 구분한다. - Package의 마지막 세그먼트는 inf.dto를 쓴다.
클래스 - 구성	<ul style="list-style-type: none"> - 입력 IO: 시스템구분 + 일련번호2자리 + 타겟거래코드 + InDto - 출력 IO: 시스템구분 + 일련번호2자리 + 타겟거래코드 + OutDto - 입력 반복 IO: 입력IO이름 + Sub + 반복 DTO명 - 출력 반복 IO: 출력IO이름 + Sub + 반복 DTO명
클래스 - 적용사례	<ul style="list-style-type: none"> - 입력 IO: EAI01SYER3101A400InDto - 출력 IO: EAI01SYER3101A400OutDto - 입력 반복 IO: EAI01SYER3101A400InDtoSubGrid00 - 출력 반복 IO: EAI01SYER3101A400OutDtoSubGrid00
클래스 - 기본원칙	<ul style="list-style-type: none"> - 시스템 구분, 일련번호 2자리, 타겟거래코드 및 Suffix로 이루어진다. - 입력에 사용되는 IO는 Suffix 를 'InDto' 으로 사용한다. - 출력에 사용되는 IO는 Suffix 를 'OutDto' 으로 사용한다. - 입/출력 IO의 서브 IO는 Suffix 를 [Sub + 반복 IO명]로 사용한다. - 파일의 확장자는 omm 을 사용한다.
클래스 - 부가정보	- IO는 소스코드 생성 방식으로 생성 되므로 사용자가 부가 정보를 입력하지 않는다.
클래스 - 작성 예	소스코드를 직접 생성하지 않는다.
메소드 - 구성	IO는 자바 소스코드가 자동생성 되므로 개발자가 직접 메소드를 구성할 필요가 없다.

Component를 위한 IO 작성

Component 메소드의 입력은 기본타입, 일반 객체를 자유롭게 사용할 수 있다. Component 메소드의 IO가 필요한 경우에는 Controller, Service의 IO와 DBIO의 IO를 재사용한다. 재사용이 어려운 경우 다음을 참고하여 생성하여 사용한다.

[표 2.13] Component IO 명명 규칙

구분	설명
Package - 구성	L1코드 + . + L2코드 + . + online 또는 batch + . + component.dto
Package - 적용사례	bxcn.sample.online.component.dto
Package - 기본원칙	<ul style="list-style-type: none"> - Package 이름 작성은 모두 소문자로 한다. - 분류는 "."로 구분한다. - Package의 마지막 세그먼트는 component.dto를 쓴다.
클래스 - 구성	<ul style="list-style-type: none"> - 입력 IO: Component 이름 + (식별가능한 표준단어) + 일련번호2자리 + InDto - 출력 IO: Component 이름 + (식별가능한 표준단어) + 일련번호2자리 + OutDto - 입력 반복 IO: Component 이름 + List + 일련번호2자리 + InDto - 출력 반복 IO: Component 이름 + List + 일련번호2자리 + OutDto - 공통 IO : Component 이름 + Dto
클래스 - 적용사례	<ul style="list-style-type: none"> - 입력 IO: SmpEmployeeComponent01InDto - 출력 IO: SmpEmployeeComponent01OutDto - 입력 반복 IO: SmpEmployeeComponentList01InDto - 출력 반복 IO: SmpEmployeeComponentList01OutDto - 공통 IO: SmpEmployeeComponentDto
클래스 - 기본원칙	<ul style="list-style-type: none"> - Component의 입출력에 사용되는 IO는 Component이름, 일련번호, 입출력 Suffix 로 이루어진다. - 입력에 사용되는 IO는 Suffix 를 'InDto' 으로 사용한다. - 출력에 사용되는 IO는 Suffix 를 'OutDto' 으로 사용한다. - 반복 IO는 Component 이름 뒤에 List로 사용한다. - 작성된 반복 IO는 Component 입출력 IO에 포함될 수 있다. - 공통 IO는 해당 Component 전체 도메인에 해당하는 IO로, 해당 Component 어디에서든 사용 할 수 있다. - 파일의 확장자는 omm 을 사용한다.
클래스 - 부가정보	- IO는 소스코드 생성 방식으로 생성 되므로 사용자가 부가 정보를 입력하지 않는다.
클래스 - 작성 예	소스코드를 직접 생성하지 않는다.
메소드 - 구성	IO는 자바 소스코드가 자동생성 되므로 개발자가 직접 메소드를 구성할 필요가 없다.

2.4.8 Variables / Constants

Controller, Service, Component 클래스의 멤버 변수로 사용되는 변수 명명 규칙에 대해 설명한다. 변수 및 상수 이름의 제한은 다음의 항목에 적용한다.

(1) 멤버 변수 선언

구분	설명
멤버변수 선언 - 구성	클래스 이름에서 첫 글자는 소문자
멤버변수 선언 - 적용 사례	mSmpEmpInfMng
멤버변수 선언 - 기본 원칙	<ul style="list-style-type: none"> - 클래스 이름에서 첫 글자는 소문자로 표기하고 나머지는 클래스 이름과 동일하게 사용한다. - 클래스 본문에 작성한다(메소드 본문에 선언하지 않는다). - Nested Class에 작성하지 않는다. - get/set 메소드를 작성하지 않는다. - private 로 접근 제한자를 선언 한다. - final 및 static 으로 선언하지 않는다. - new 로 직접 객체를 생성하지 않는다.
멤버변수 선언 - 작성 예	<pre>private SmpEmployeeComponent smpEmployeeComponent;</pre>

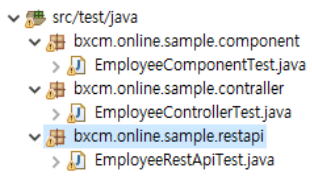
(2) 상수

구분	설명
상수 - 구성	표준단어 + (_ + 표준단어)*
상수 - 적용 사례	<ul style="list-style-type: none"> - MAX_NUM - 시스템코드_CODE_NAME
상수 - 기본 원칙	<ul style="list-style-type: none"> - 상수는 static final로 선언 되며 이름은 모두 대문자로 이루어진다(외부 공개가 필요한 상수는 public 키워드를 함께 사용). - 각 단어는 _(언더바)로 구분한다. - 상수 길이는 50글자 이내로 작성한다. - 초기화는 static 블록에서 이루어진다.
상수 - 작성 예	<pre>public static final int MAX_NUM;</pre>

구분	설명
	<pre>public static final String 시스템코드_CODE_NAME; static { MAX_NUM= 100; CODE_NAME= "BXM"; }</pre>

2.4.9 테스트 케이스

[표 2.14] 테스트 케이스 규칙

구분	설명
폴더명 - 구성	bxcn.sample.online.component
파일명 - 구성	Component(Class)이름 +Test
파일명 - 적용사례	SmpEmployeeComponentTest.java SmpEmployeeControllerTest.java
메소드명 - 구성	test + 메소드명
메소드명 - 적용사례	testGetEmployee() testGetEmployee01(), testGetEmployee02() testGetEmployeeFail(), testGetEmployeeSuccess()
파일명, 메소드명 - 기본 원칙	<p>package는 테스트를 하려는 서비스, 컴퍼넌트와 동일하게 작성한다.</p> <p>클래스명은 뒤에 Test를 붙여 작성한다.</p> <p>테스트 메소드명은 테스트하려는 메소드 앞에 test 를 붙여 작성한다.</p> <p>하나의 메소드에 대해 여러 개의 테스트 케이스를 작성할 경우에는 테스트 메소드에 일련번호 혹은 식별할 수 있는 단어를 조합한다.</p>
파일명 - 작성 예	 <p>[그림 2.1] 테스트 케이스 생성 모습</p>

2.4.10 지역변수

메소드 내부에서 사용하는 지역변수의 이름은 따로 표준을 제공하지 않는다. 자유롭게 선언하여 사용하되 가급적 의미를 쉽게 파악할 수 있는 이름을 사용한다.

2.4.11 명명규칙 예제

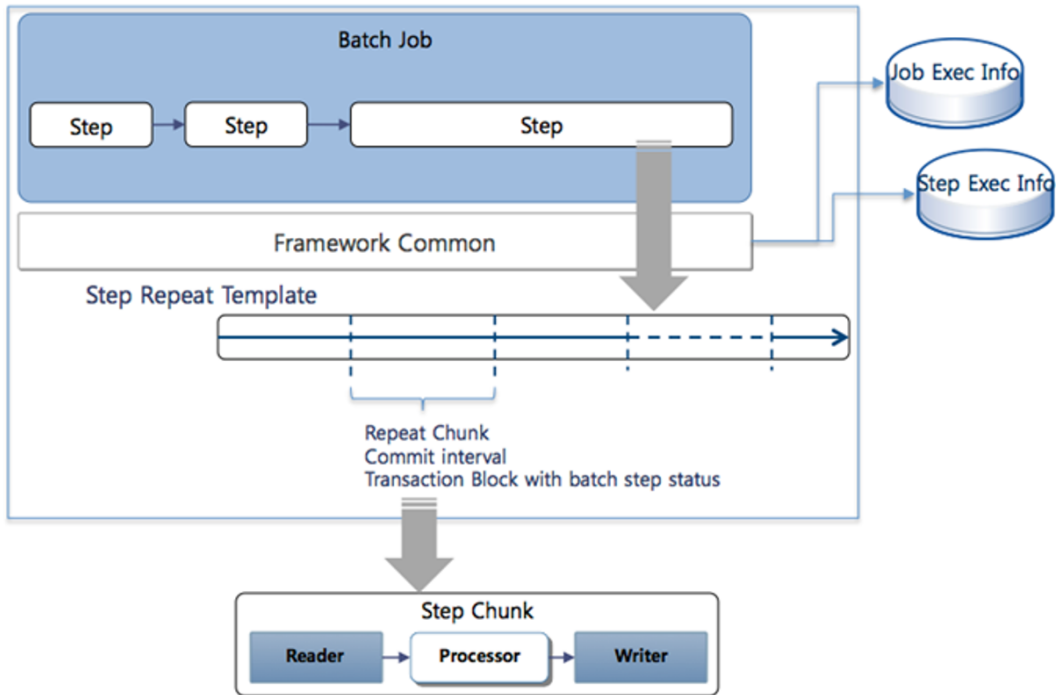
본 장에서 가이드 한 명명규칙을 지키면 다음 표와 같은 이름의 리소스가 생성 되어야 한다.

구분	설명
프로젝트	<ul style="list-style-type: none"> - 온라인 프로젝트 : bxcn-sample-online - 배치 프로젝트 : bxcn-sample-batch
Controller 클래스 이름 및 Package	<ul style="list-style-type: none"> - Package : bxcn.sample.online.controller - 클래스 : SmpEmployeeController
Controller 클래스 이름 및 Package IO 이름 및 Package	<ul style="list-style-type: none"> - Package : bxcn.sample.online.controller.dto - Input IO : SmpEmployeeController01InDto - Output IO : SmpEmployeeController01OutDto
Service 클래스 이름 및 Package	<ul style="list-style-type: none"> - Package : bxcn.sample.online.service - 클래스 : SmpEmployeeService
Controller 클래스 이름 및 Package IO 이름 및 Package	<ul style="list-style-type: none"> - Package : bxcn.sample.online.service.dto - Input IO : SmpEmployeeService01InDto - Output IO : SmpEmployeeService01OutDto
Component 이름 및 Package	<ul style="list-style-type: none"> - Package : bxcn.sample.online.component - 클래스 : SmpEmployeeComponent
Component IO 이름 및 Package	<ul style="list-style-type: none"> - Package : bxcn.sample.online.component.dto - Input IO : SmpEmployeeComponent01InDto - Output IO : SmpEmployeeComponent01OutDto
DBIO 이름 및 Package	<ul style="list-style-type: none"> - Package : bxcn.sample.online.dbio - 클래스 : DSmpEmpTst001
DBIO IO 이름 및 Package	<ul style="list-style-type: none"> - Package : bxcn.sample.online.dbio.dto - Input IO : DSmpEmpTst001selectPage01InDto - Output IO : DSmpEmpTst001selectPage01OutDto

2.5 배치 어플리케이션 명명규칙

2.5.1 배치작업 처리 모델

다음은 기본적인 유형의 배치작업 처리 모델이다.



[그림 2.2] 배치작업 처리 모델

2.5.2 배치 어플리케이션

BXCM은 배치를 위한 어플리케이션 단위는 관리 주체가 다르고, 어플리케이션 간 기능 호출을 최소화 할 수 있는 단위로 구분 한다. 어플리케이션은 Studio 의 '새로운 BX 클라우드 프로젝트' 메뉴를 통해 생성한다.

[표 2.15] 배치 어플리케이션 명명 규칙

구분	설명
구성	L1코드 + '-' + L2코드 + '-' + batch
적용사례	bxcn-sample-batch
기본원칙	<ol style="list-style-type: none"> 어플리케이션 이름 작성은 다음과 같은 표기법을 따른다. L1코드,L2코드(영문명)의 소문자로 표기 단어와 단어 사이에 '-' 로 구분 어플리케이션 이름은 중복되어서는 안되므로 전체 업무를 통틀어 유일한 이름을 갖도록 작성한다.

2.5.3 Job

배치 업무 처리 프로세스의 논리적 단위이며, 복수의 Step으로 구성되고, 배치 업무 실행 단위가 된다. Job의 이름은 다음과 같이 작성한다.

[표 2.16] Job 명명 규칙

구분	설명
구성	L2코드(영문명) + "-" (표준단어)* - 배치 Job을 식별 할 수 있는 단어의 조합으로 소문자로 구성한다.
적용사례	sample-account-tax-sum sample-total-proc
기본원칙	- 배치 Job 명을 작성은 다음과 같은 표기법을 따른다. - 단어는 소문자와 '-' 로 조합한다. - L2코드(영문명)과 표준단어들의 조합으로 구성한다.
작성 예	<pre> <job id="sample-account-tax-sum"> ... </job> </pre>

2.5.4 Step

Step은 배치의 최소 실행 업무 단위이다. 하나의 Job 에 1-N개의 Step 이 존재하며, Step 은 순차적으로 실행 된다. Step 을 순차적으로 실행하지 않고 병렬로 처리 할 경우 Step의 종류인 Split을 사용한다. Step과 Split의 이름은 다음과 같이 작성한다.

구분	설명
구성	JOB ID + '-' + Step일련번호 1자리 + Split일련번호2자리 Step만 있는 경우 100, 200 순으로 증가한다.
적용사례	sample-account-tax-sum-100 sample-account-tax-sum-101
기본원칙	- Step 일련번호1자리: 배치작업의 Main Step 을 일련번호 순차부여(1,2,3...) - Split일련번호2자리 - Main Step이 Split 되지 않으면 "00"으로 고정 - Split 되는 경우 일련번호 순차부여(01,02...)
작성 예	- 단일 Step <pre> <job id="sample-account-tax-sum"> <step id="sample-account-tax-sum-100"> ... </step> </job> </pre>

구분	설명
	<p>– 복수 Step</p> <pre> <job id="sample-account-tax-sum"> <step id="sample-account-tax-sum-100" next="sample-account-tax-sum-200"> ... </step> </job> </pre> <p>– Split</p> <pre> <job id="sample-account-tax-sum" restartable="true"> <step id="sample-account-tax-sum-100" next="sample-account-tax-sum-200"> <flow> <step id="sample-account-tax-sum-101"> ... </step> </flow> <flow> <step id="sample-account-tax-sum-102"> ... </step> </flow> </step> <step id="sample-account-tax-sum-200"> ... </step> </job> </pre>

2.5.5 Job Configuration File

Job의 Configuration을 설정하는 파일의 이름은 다음과 같이 작성한다.

[표 2.17] Job Configuration File 규칙

구분	설명
구성	JOB ID + .xml
적용사례	sample-account-tax-sum.xml sample-total-proc.xml
기본원칙	<p>– 프로젝트의 src/main/resources/jobs 폴더에 생성한다.</p> <p>– 파일의 확장자는 xml를 사용한다.</p> <p>– 하나의 Configuration 파일에 하나의 JOB이 존재한다.</p>

2.6 기타 명명 규칙

2.6.1 거래 ID

거래 ID는 다음과 같이 작성한다.

[표 2.18] 온라인 거래코드

구분	설명
구성	L2코드(영문명) + "_" (식별가능한 표준단어 조합)* + "_" 일련번호 2자리
적용사례	SMP_EMPLOYEE_01
기본원칙	<ul style="list-style-type: none">- 거래코드는 L2코드(영문명)과 식별 가능한 표준단어와 일련번호로 구성된다.- Controller / RestController에 정의된 URL과 매핑 된다.- 대문자로 구성한다.

2.6.2 로그 파일

로그 파일의 경우에는 클라우드 환경(Kubernetes)에서는 별도의 로그 파일을 생성하지 않는 경우가 있다. 별도의 로그 파일의 생성이 필요하다면 다음과 같은 규칙으로 로그 파일을 생성하며, 인스턴스 별로 생성이 된다.

참고로 거래 로깅(logging)을 위한 기본적인 로그는 서비스 로그 테이블에 저장되며, 웹어드민 화면을 통하여 확인할 수 있다.

[표 2.19] 온라인 로깅

구분	설명
구성	온라인 로그 prefix + 노드명 + 인스턴스명
적용사례	app_DFT1_online1.log
기본원칙	<ul style="list-style-type: none">- 온라인 로그 prefix, 노드명, 인스턴스명으로 이루어진다.- 확장자는 log 를 사용한다.- 로그 파일의 위치는 다음과 같다 \${로그설정위치}/online

2.6.3 배치 작업 ID

배치 작업 ID는 다음과 같이 작성한다.

[표 2.20] 배치 작업 ID

구분	설명
구성	JOB ID
적용사례	sample-account-tax-sum sample-total-proc
기본원칙	- 배치 작업 ID는 JOB ID와 동일하게 구성된다.

2.6.4 배치 로그 파일

배치 로그 파일의 이름은 다음과 같이 작성 된다. 업무 개발자가 직접 작성할 필요가 없으며 배치가 수행 되면 생성일 시 별로 생성 된다.

[표 2.21] 배치 로깅

구분	설명
구성	Job ID + _ + 일시14자리
적용사례	sample-account-tax-sum_20230806165255.log sample-account-tax-sum_20230807102740.log
기본원칙	- JOB ID와 실행 일시(14자리)로 이루어진다. - 확장자는 log 를 사용한다. - 로그 파일의 위치는 다음과 같다 \${로그설정위치}/batch/\${yyyyMMdd}/