Wkit API Reference

컴포넌트 라이프사이클 관리, 이벤트 바인딩, 3D 리소스 정리를 위한 핵심 유틸리티.

2D 이벤트 바인딩

bindEvents(instance, customEvents)

2D 컴포넌트에 이벤트 델리게이션 바인딩.

```
/**
 * @param {Object} instance - 컴포넌트 인스턴스
 * @param {Object} customEvents - 이벤트 매핑 객체
 */
Wkit.bindEvents(this, this.customEvents);
```

customEvents 형태:

```
{
    click: {
        '.my-button': '@buttonClicked',
        '.nav-link': '@navLinkClicked'
    },
    submit: {
        'form': '@formSubmitted'
    }
}
```

내부 동작:

1. `instance.appendElement` 에 이벤트 리스너 등록
2. `event.target.closest(selector)` 로 이벤트 타겟 확인
3. 매칭 시 `Weventbus.emit(triggerEvent, { event, targetInstance })` 호출

---

removeCustomEvents(instance, customEvents)

바인딩된 2D 이벤트 제거.

```
Wkit.removeCustomEvents(this, this.customEvents);
```

참조: `instance.userHandlerList` 에 저장된 핸들러 참조를 사용하여 제거.

3D 이벤트 바인딩

initThreeRaycasting(target, eventName)

3D 씬에 Raycasting 이벤트 초기화.

```
/**
 * @param {HTMLElement} target - 이벤트 수신 요소 (Canvas 또는 컨테이너)
 * @param {string} eventName - 브라우저 이벤트 ('click', 'mousemove' 등)
 * @returns {Function} - 생성된 이벤트 핸들러 (정리용)
 */
```

```
const onRaycasting = Wkit.initThreeRaycasting(
    document.getElementById('threeCanvas'),
    'click'
);
```

내부 동작:

1. `THREE.Raycaster` 로 마우스 좌표 → 3D 교차점 계산
2. 교차 객체의 부모 체인에서 `eventListener` 속성 탐색
3. 찾으면 `eventListener[eventType](event)` 호출

---

bind3DEvents(instance, customEvents)

3D 컴포넌트에 이벤트 핸들러 등록.

```
/**
 * @param {Object} instance - 3D 컴포넌트 인스턴스
 * @param {Object} customEvents - 이벤트 매핑
 */
Wkit.bind3DEvents(this, this.customEvents);
```

customEvents 형태 (3D):

```
{
    click: '@meshClicked',
    mousemove: '@meshHovered'
}
```

내부 동작:

- `instance.appendElement.eventListener[browserEvent]` 에 핸들러 등록
- Raycasting 시스템이 이 핸들러를 호출

3D 리소스 정리

dispose3DTree(rootContainer)

Three.js 객체 트리 전체 정리.

```
/**
 * @param {THREE.Object3D} rootContainer - 정리할 루트 객체
 */
Wkit.dispose3DTree(this.appendElement);
```

정리 대상:

항목	처리
geometry	`dispose()`
material	텍스처 슬롯 정리 후 `dispose()`
textures	13개 슬롯 순회하여 `dispose()`
eventListener	속성 해제

userData	빈 객체로 초기화
parent	`parent.remove(rootContainer)`

---

clearSceneBackground(scene)

씬 배경 정리.

```
Wkit.clearSceneBackground(wemb.threeElements.scene);
```

---

disposeAllThreeResources(page)

페이지 내 모든 3D 리소스 일괄 정리.

```
Wkit.disposeAllThreeResources(this.page);
```

정리 순서:

1. `instance.subscriptions` → GlobalDataPublisher 구독 해제
2. `instance.appendElement` → `dispose3DTree` 호출
3. `scene.background` → 정리

> ***Note:*** *인스턴스 속성(`customEvents`, `datasetInfo` 등)의 수동 null 처리는 수행하지 않는다. 인스턴스 자체가 GC될 때 속성도 함께 수거되며, 외부에서 속성을 null 처리하면 컴포넌트 내부 정리 로직(`_onViewerDestroy`)의 전제 조건을 깨뜨릴 수 있다. 상세: INSTANCE_LIFECYCLE_GC.md 부록 A*

헬퍼 함수

makeIterator(page, ...layerList)

페이지의 컴포넌트 인스턴스 이터레이터 생성.

```
/**
 * @param {Object} page - 페이지 객체
 * @param {...string} layerList - 레이어 이름 ('masterLayer', 'twoLayer', 'threeLayer')
 * @returns {Generator} - 인스턴스 이터레이터
 */
const iter = Wkit.makeIterator(this.page, 'twoLayer', 'threeLayer');

// 모든 2D, 3D 인스턴스 순회
for (const instance of iter) {
    console.log(instance.name);
}
```

레이어 → 맵 매핑:

레이어	인스턴스 맵
masterLayer	`componentInstanceListMap`
twoLayer	`componentInstanceListMap`
threeLayer	`_appendElementListMap`

---

getInstanceByName(instanceName, iter)

이름으로 인스턴스 검색.

```
const instance = Wkit.getInstanceByName(
    'MyComponent',
    Wkit.makeIterator(this.page)
);
```

---

getInstanceById(targetId, iter)

ID로 인스턴스 검색.

```
const instance = Wkit.getInstanceById(
    'component-123',
    Wkit.makeIterator(this.page)
);
```

---

fetchData(page, datasetName, param)

데이터 서비스 호출 (Promise 래핑).

```
/**
 * @param {Object} page - 페이지 객체
 * @param {string} datasetName - 데이터셋 이름
 * @param {Object} param - 요청 파라미터
 * @returns {Promise<any>} - 응답 데이터
 */
const data = await Wkit.fetchData(this.page, 'myapi', { id: 123 });
```

내부 동작:

```
page.dataService
    .call(datasetName, { param })
    .on('success', resolve)
    .on('error', reject);
```

---

emitEvent(eventName, targetInstance)

코드에서 직접 이벤트 발생.

```
Wkit.emitEvent('@myCustomEvent', this);
```

---

withSelector(element, selector, fn)

안전한 querySelector + 함수 실행.

```
/**
 * @param {HTMLElement} element - 부모 요소
 * @param {string} selector - CSS 선택자
 * @param {Function} fn - 요소에 실행할 함수
 * @returns {any|null} - 함수 반환값 또는 null
```

```
 */
Wkit.withSelector(this.appendElement, '.status', (el) => {
    el.textContent = 'Active';
});
```

이벤트 버스 핸들러 관리

onEventBusHandlers(eventBusHandlers)

여러 이벤트 버스 핸들러 일괄 등록.

```
/**
 * @param {Object} eventBusHandlers - { eventName: handler } 객체
 */
Wkit.onEventBusHandlers({
    '@buttonClicked': (data) => this.handleButtonClick(data),
    '@formSubmitted': (data) => this.handleFormSubmit(data)
});
```

---

offEventBusHandlers(eventBusHandlers)

등록된 핸들러 일괄 해제.

```
Wkit.offEventBusHandlers(this.eventBusHandlers);
```

스키마 헬퍼 (참고용)

Default JS 템플릿의 기본 구조 예시를 반환합니다.

메서드	반환값	용도
`getGlobalMappingSchema()`	`{ topic, datasetInfo }`	globalDataMappings 형태
`getCustomEventsSchema()`	`{ eventType: { selector: trigger } }`	2D customEvents 형태
`getCustomEventsSchemaFor3D()`	`{ eventType: trigger }`	3D customEvents 형태
`getSubscriptionSchema()`	`{ topic: [methods] }`	subscriptions 형태

관련 문서

- DEFAULT_JS_NAMING.md - 필수 네이밍 규칙
- WEVENTBUS_API.md - 이벤트 버스
- GLOBAL_DATA_PUBLISHER_API.md - 전역 데이터