BITFOCUS

Companion

한국어 사용 설명서

V5.1.1

2026.10 · 공식 v5.1 사용자 가이드 기준

목차

1 설치와 서버 시작 1

1.1 설치 방식과 설치 방법 1
1.2 Companion Pi 3
1.3 Pi 선택 3
1.4 CompanionPi 이미지 사용 4
1.5 Pi 보안 설정 5
1.6 수동 설치 6
1.7 Headless 구성 7
1.8 Companion 업데이트 8
1.9 실행 전 준비 8
1.10 서버 시작 10
1.11 서버 구성 11
1.12 설정 항목 참조 13
1.13 고급 설정 16

2 설정 화면 사용 17

2.1 사용자 화면 17
2.2 연결 20
2.3 버튼 25
2.4 버튼 만들기 30
2.5 버튼 스타일 37
2.6 텍스트 요소 45
2.7 이미지 요소 48
2.8 게이지 요소 50
2.9 그룹과 참조 52
2.10 캔버스와 버튼 표시 정보 55
2.11 액션 57
2.12 피드백 60
2.13 단계 66
2.14 페이지 67
2.15 액션 기록기 69
2.16 제어 장치 70
2.17 트리거 73
2.18 변수 76

2.19 이미지 라이브러리 83
2.20 모듈 85
2.21 설정 87
2.22 가져오기 / 내보내기 94
2.23 로그 96
3 표현식 97
3.1 연산자 102
3.2 함수 106
3.3 고급 표현식 120
4 원격 제어 124
4.1 REST API 124
4.2 REST API 인증 124
4.3 연결 REST API 126
4.4 제어 장치 REST API 135
4.5 TCP / UDP 제어 140
4.6 HTTP 원격 제어 142
4.7 OSC 제어 146
4.8 Prometheus로 모니터링 147
4.9 Artnet / DMX 제어 148
4.10 EmberPlus 제어 149
4.11 RossTalk 제어 151
4.12 Satellite 프로토콜 152
5 웹 버튼과 에뮬레이터 153
5.1 에뮬레이터 153
5.2 웹 버튼 154
6 지원 제어 장치 156
6.1 Companion Satellite 158
7 보안 159
부록 · 원문과 편집 기준 160

1 설치와 서버 시작

1.1 설치 방식과 설치 방법

다운로드

Companion은 웹사이트에서 다운로드할 수 있습니다.
웹사이트에서는 다음과 같은 배포 버전을 제공합니다.

정식 버전(Stable)

Companion을 처음 사용하거나 어떤 버전을 사용할지 모르겠다면 정식 버전으로 시작하세요.

오류가 없도록 추가 검증을 거친 버전입니다. 몇 달마다 업데이트되며, 일반적으로 새 기능은 베타 버전보다 늦게 제공됩니다.

베타 버전(Beta)

아직 정식 버전에 포함되지 않은 기능이 필요하고 가끔 발생하는 오류를 감수할 수 있다면 베타 버전을 사용하세요.
오류가 충분히 해결되었다고 판단되면 이 버전을 바탕으로 새 정식 버전을 출시합니다.

현재 개발 중인 버전입니다. 오류가 생길 수 있지만 새 기능이 자주 추가됩니다.
대체로 일상적인 사용에 충분히 안정적이지만, 항상 그런 것은 아닙니다.

설치

Mac

참고

5.0부터 macOS 13.5 이상을 지원합니다.

다운로드한 파일의 압축을 풀고 Companion.app을 응용 프로그램(Applications) 폴더로 드래그합니다.

Windows (x64)

참고

3.0부터 Windows 10 이상을 지원합니다.

다운로드한 파일을 설치한 뒤 실행합니다.

Raspberry Pi

CompanionPi 설정 안내에서 제공하는 CompanionPi 이미지 사용을 권장합니다.

Pi에 직접 설치하려면 아래의 Linux 설치 방법을 따릅니다.

Linux (x64 및 arm64)

Linux 다운로드에는 **데스크톱(desktop)** 빌드와 **화면 없는 서버용(headless)** 빌드가 모두 포함되어 있습니다.
Companion을 실행할 방식에 맞게 선택합니다. 어느 쪽을 선택하든 다운로드에 포함된 README에서 시스템 의존성과 USB 권한(udev) 설정 방법을 확인합니다.

데스크톱(그래픽 화면 사용)

다운로드한 파일의 압축을 풀고 companion-launcher를 실행합니다.

Linux에서 Stream Deck과 다른 USB 제어 장치를 사용하려면 udev 규칙을 설치해야 합니다.
설정이 필요하면 Companion 화면에 안내가 표시되며, 데스크톱에서는 Companion이 직접 규칙을 적용할 수 있습니다.
자세한 내용은 README를 참조하세요.

"Using GTK 2/3 and GTK 4 in the same process is not supported" 오류로 실행에 실패하면
--gtk-version=3 인수를 추가합니다.

Headless(서버 / 그래픽 화면 없음)

화면 없는 환경이나 서버에 설치할 때는 설치 스크립트 사용을 권장합니다. 이 스크립트는 CompanionPi와
유사한 환경을 구성하며 시스템 의존성, companion 사용자와 그룹, udev 규칙을 자동으로 설정합니다.
자세한 내용은 수동 설치를 참조하세요.

다운로드에는 별도로 실행할 수 있는 headless 빌드도 포함되어 있습니다. 함께 제공되는 companion_headless.sh
스크립트로 실행할 수 있습니다. 이 방식도 지원하지만 의존성·그룹·udev 설정을 모두 직접 해야 하므로 권장하지 않습니다.
다운로드에 포함된 README를 참조하세요.

Docker

Linux 배포를 간소화할 수 있는 Docker 이미지를 GitHub Container Registry에 제공합니다.
linux/amd64와 linux/arm64용 이미지가 제공됩니다.

```
docker run -d --name companion \
  -p 8000:8000 -p 16622:16622 -p 16623:16623 \
  -v companion-config:/companion \
  ghcr.io/bitfocus/companion/companion:latest
```

docker-compose를 사용할 수도 있습니다.

```
services:
  companion:
    image: ghcr.io/bitfocus/companion/companion:latest
    restart: unless-stopped
    ports:
      - '8000:8000' # Admin UI
      - '16622:16622' # Companion Satellite (TCP)
      - '16623:16623' # Companion Satellite (WS)
    volumes:
      - companion-config:/companion
volumes:
  companion-config:
```

설정이 유지되도록 반드시 /companion에 볼륨을 연결하세요.

Companion은 여러 수신 포트를 사용합니다. 아래 포트 외에도 일부 모듈은 자체 서버를 실행하여 외부에서 들어오는 연결을 받습니다. Docker의 네트워크 모드를 정할 때 이를 고려해야 합니다.

포트	용도
8000	관리 화면 및 웹 API
16622	Companion Satellite (TCP)
16623	Companion Satellite (WS)

구성

실행 옵션(서버 구성의 관리 포트, 로깅, syslog, 보안 옵션 등)은 설정 볼륨 안의 config.yaml 파일(/companion/config.yaml)에서 읽습니다. 처음 실행하면 설명 주석이 포함된 파일이 자동 생성됩니다. 파일을 수정한 뒤 컨테이너를 다시 시작하면 변경 사항이 적용됩니다. Headless / CompanionPi 설치에서도 같은 파일과 도구를 사용합니다. 전체 옵션은 설정 항목 참조에서 확인할 수 있습니다.

COMPANION_ADMIN_PORT(기본값 8000)는 파일을 처음 만들 때 반영되며 컨테이너 상태 검사에도 사용됩니다. config.yaml에서 adminPort를 변경했다면 공개한 포트 매핑도 같은 값으로 변경합니다.

USB 패스스루

현재 지원하지 않습니다. 사용 중인 USB 라이브러리가 Docker에서 제대로 작동하지 않습니다.

원격 USB

다른 컴퓨터의 Stream Deck이나 제어 장치를 Companion에 연결하려면 Companion Satellite를 사용할 수 있습니다. 이를 사용하려면 TCP 포트 16622를 포워딩해야 합니다.

1.2 Companion Pi

Raspberry Pi에서 Companion을 실행할 수 있도록 미리 구성된 이미지를 제공합니다. 원하는 SBC(싱글 보드 컴퓨터)나 다른 Linux 컴퓨터에 Companion을 설치할 수 있는 도구도 제공합니다.

시작하기

Pi에서 Companion을 실행하는 방법은 크게 두 가지입니다.

1. **CompanionPi 이미지 사용** - 가장 간단한 방법입니다. 미리 구성된 이미지를 다운로드하여 SD 카드에 기록합니다.
2. **수동 설치** - 기존 Linux 시스템에 Companion을 설치하려는 숙련된 사용자를 위한 방법입니다.

시작하기 전에 Pi 선택 안내를 확인하세요.

1.3 Pi 선택

지원 모델

공식적으로 지원하는 모델은 **Raspberry Pi 4 및 5**와 이를 기반으로 한 400, Compute Module 4 등의 모델입니다.

이전 모델에서도 실행할 수는 있지만 **권장하지 않으며 지원하지 않습니다**. 사용하려면 발생할 수 있는 문제를 직접 감수해야 하며, 커뮤니티의 도움을 받기 어려울 수 있다는 점을 이해해야 합니다.

구형 Pi 4의 펌웨어

구형 Pi 4에 Companion을 새로 설치한다면 최신 EEPROM/펌웨어 업데이트가 적용되어 있는지 확인하세요(관련 정보).

2019년 10월 말 업데이트에서는 SPI EEPROM과 VLI USB 컨트롤러 칩의 업데이트 방식이 통합되었습니다.
최신 업데이트를 설치하면 향후 네트워크에 연결된 장치나 외부 USB 저장 장치에서 Raspberry Pi를 부팅할 수 있게 됩니다.
또한 VLI 펌웨어가 업데이트되어 소비 전력이 줄고 작동 온도가 최대 3-4°C 낮아집니다.

이전 모델을 지원하지 않는 이유

4 이전 모델에서는 안정성 문제가 확인되어 지원하지 않습니다. 다음과 같은 여러 요인이 원인으로 추정되며,
요인끼리 서로 영향을 줄 수도 있습니다.

- 전원 출력 능력(예: Stream Deck에 전원을 공급하는 경우)
- 공유 USB 버스를 사용하는 이더넷과 Stream Deck 간의 간섭
- 최대 RAM 용량
- CPU 성능

자세한 내용은 이슈 #313을 참조하세요. 이에 따라 현재 개발은 Raspberry Pi 4 이상 시스템에 집중하고 있습니다.

1.4 CompanionPi 이미지 사용

Raspberry Pi에서 Companion을 실행하는 가장 간단한 방법은 미리 구성된 CompanionPi 이미지를 사용하는 것입니다.

이미지 다운로드

Bitfocus 웹사이트에서 이미지를 다운로드할 수 있습니다.

MicroSD 카드 선택

속도가 빠른 MicroSD 카드가 필요합니다. 널리 사용되는 제품으로 SanDisk Extreme PRO 32GB 카드가
있습니다(Amazon US 제품 링크).

언제나 최고의 선택은 아니지만, 충분히 빠르고 성능이 좋은 microSD 카드입니다. **UHS 1 이상** 등급의 제품을 선택하세요.
Raspberry Pi를 지나치게 느리지 않게 사용하려면 이 정도의 읽기·쓰기 속도가 필요합니다.

이미지 기록

빠른 microSD 카드를 준비했다면 이미지를 카드에 기록할 도구가 필요합니다. 널리 사용되는 도구로 Windows, Mac,
Linux용 Raspberry Pi Imager가 있습니다. 사용하기 쉽고 이미지를 기록하면서 SSH도 설정할 수 있습니다.

참고 첫 부팅

처음 부팅하면 관리 화면이 열리기까지 몇 분이 걸립니다. 운영체제가 SD 카드 용량에 맞게 영역을 확장하며,
이 과정에서 한 번 더 재부팅합니다.

관리 화면 접속

CompanionPi 이미지로 Raspberry Pi를 실행했다면 Pi의 IP 주소를 확인해야 합니다. 다음과 같은 방법이 있습니다.

- **Python 스크립트**: 부팅할 때마다 IP 주소를 이메일로 보내는 사용자 작성 스크립트를 사용할 수 있습니다. 부팅 시 인터넷 연결이 필요합니다. GitHub 스크립트를 참조하세요.
- **고정 IP**: Pi에 고정 IP 주소를 설정합니다. Raspberry Pi를 항상 같은 장비에 연결한다면 좋은 방법입니다. 3.1.1 이하용 The Pi Hut 안내 또는 3.1.2 이상용 안내를 참조하세요.
- **LCD 디스플레이**: LCD를 연결하여 현재 IP 주소를 표시합니다. 직접 제작하는 재미가 있는 방법입니다. PiMyLifeUp 예제를 참조하세요.

IP 주소를 확인했다면 해당 주소의 포트 8000으로 Companion 관리 화면에 접속합니다.
예를 들어 http://192.168.1.3:8000을 사용합니다.

1.5 Pi 보안 설정

SSH 사용자 설정

Raspberry Pi의 2022년 4월 변경 사항에 따라 CompanionPi 이미지에서는 사용자를 설정하기 전까지 SSH 접속을 허용하지 않습니다. 화면을 연결했다면 사용자 설정 안내가 표시됩니다. 또는 첫 부팅 때 설정 파일로 사용자를 생성할 수 있습니다.

Raspberry Pi Imager 사용

Raspberry Pi Imager를 사용하면 이미지를 기록하기 전에 사용자 이름과 비밀번호를 설정할 수 있습니다.

첫 부팅 시 사용자 생성

1. SD 카드의 최상위 디렉터리에 userconf.txt 파일을 만듭니다.
2. 다음 명령을 실행하여 비밀번호의 해시를 생성합니다.

```
openssl passwd -6 <your-password>
```

3. userconf.txt 파일에 다음 한 줄을 추가합니다.

```
<username>:<password-hash>
```

비밀번호 해시에는 앞 단계의 출력값을 사용합니다.

도움말

사용자 이름으로 companion을 사용하면 안 됩니다. Companion을 실행하는 사용자와 충돌합니다.

공격자가 사용자 이름을 쉽게 추측하지 못하도록 pi도 사용하지 않는 것을 권장합니다.

권장 보안 설정

다음과 같은 보안 설정도 권장합니다.

- sudo 사용 시 비밀번호를 요구하도록 설정
- 최신 운영체제 업데이트와 보안 수정 사항 적용
- SSH 보안 강화

이러한 권장 설정은 raspberrypi.org 웹사이트에서 확인할 수 있습니다.

Companion의 일반 보안 설정은 보안 페이지를 참조하세요.

1.6 수동 설치

주의 숙련된 사용자용

Linux에 익숙하고 문제를 직접 진단할 수 있는 사용자만 이 방법을 사용하세요. 제공 도구의 결함이 아니라면 발생한 문제를 해결하는 데 도움을 드리지 못할 수 있습니다.

요구 사항

x64 또는 ARM64 컴퓨터에서 지원하며, CPU 아키텍처에 맞는 Debian 또는 Ubuntu 기반 운영체제가 필요합니다.

Companion이 사용할 수 있는 자원을 최대한 확보하도록 **Headless 또는 서버용 설치**를 권장합니다.

준비

설치를 시작하기 전에 운영체제를 설치하고 설정해야 합니다.

이때 운영체제에 최신 업데이트가 적용되어 있는지도 확인하세요.

```
sudo apt update && sudo apt upgrade -y
```

설치

권장 설치 방법은 root 권한으로 다음 명령을 실행하는 것입니다.

```
curl https://raw.githubusercontent.com/bitfocus/companion-pi/main/install.sh | bash
```

CompanionPi 이미지와 동일한 설치 및 설정 과정을 수행합니다.

설치 프로그램이 수행하는 작업

설치 스크립트는 다음 작업을 수행합니다.

- companion 사용자 생성
- 필요한 시스템 의존성 설치
- Companion 최신 베타 빌드 다운로드
- Stream Deck과 지원되는 다른 제어 장치를 사용할 수 있도록 udev 규칙 설정
- Companion이 시스템을 종료하고 다시 시작할 수 있도록 sudo 규칙 설정
- companion-update 등의 스크립트 설치

변경되는 내용을 자세히 확인하려면 설치 스크립트를 읽어보세요.

서버 구성

관리 포트, 로깅, 보안 기능과 같은 실행 옵션은 config-tool로 관리합니다. 자세한 내용은 Headless 구성을 참조하세요.

사용자 지정

원하는 대로 설치 환경을 변경할 수 있습니다. 다만 업데이트 도구가 작동하지 않게 되거나 다음 업데이트에서 변경 사항이 덮어써지지 않도록 주의해야 합니다.

이보다 더 많은 사용자 지정이 필요하다면 이슈로 알려주세요.

1.7 Headless 구성

데스크톱 설치에서는 실행 프로그램의 설정 창에서 Companion의 실행 옵션을 변경합니다. CompanionPi와 같은 Headless 또는 서버 설치에는 실행 창이 없으므로 명령줄에서 같은 옵션을 편집합니다.

각 옵션의 기능은 서버 구성을 참조하세요. 이 페이지에서는 Headless 설치에서 옵션을 편집하는 방법을 설명합니다.

설정 편집

CompanionPi나 스크립트로 설치한 Linux 환경에서 다음 명령을 실행합니다.

```
sudo companion-config
```

반드시 **root** 권한으로 실행해야 하므로 sudo를 사용합니다. 터미널에 대화형 편집기가 열리며 옵션은 네트워크(Network), 로깅(Logging), Syslog, 보안(Security), 경로(Paths), 고급(Advanced) 페이지로 나뉩니다. 여기에서 관리 포트, 로깅, 보안 기능의 사용 여부 등을 변경할 수 있습니다.

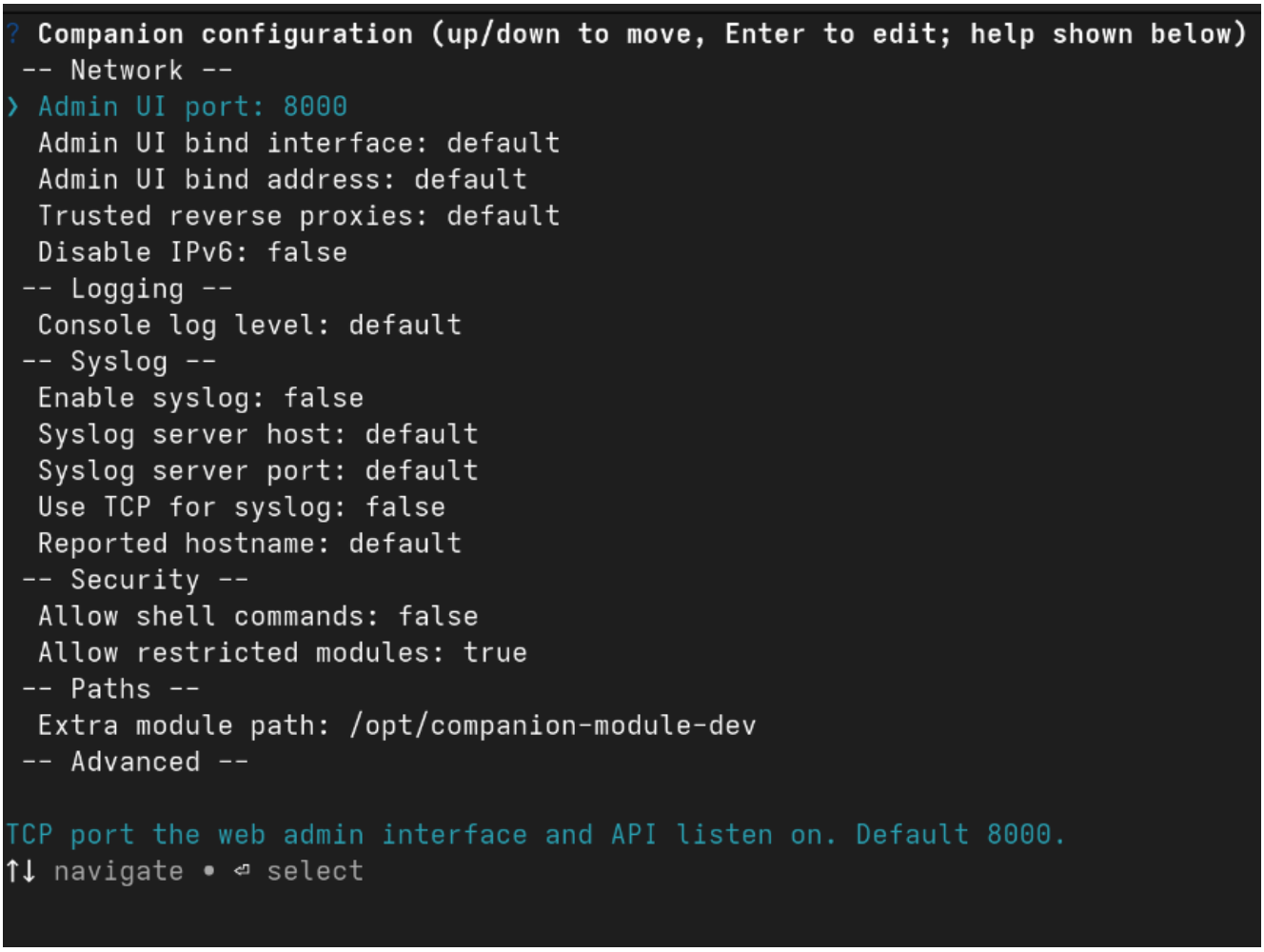


그림 1-1. 설정 도구

변경 사항을 저장하면 실행 중인 Companion 서비스가 설정을 적용하기 위해 **자동으로 다시 시작**됩니다.
Companion이 실행 중이 아니라면 다음에 시작할 때 적용됩니다.

파일 직접 편집

설정은 다음 위치의 **YAML** 파일에 저장됩니다.

```
/etc/companion/config.yaml
```

원한다면 이 파일을 직접 편집할 수 있습니다. 주석과 서식이 유지되므로 직접 편집해도 괜찮습니다.
설정을 지우고 내장 기본값으로 되돌리려면 해당 키를 삭제하거나 값을 null로 설정합니다.

sudo companion-config 대신 파일을 직접 편집했다면 변경 사항을 적용하기 위해 Companion을 직접 다시 시작해야 합니다.

```
sudo systemctl restart companion
```

1.8 Companion 업데이트

업데이트 도구 사용

Companion이 실행 중인 컴퓨터에 SSH로 접속하거나 로그인한 뒤 다음 명령으로 대화형 업데이트 도구를 실행합니다.

```
sudo companion-update
```

설치할 Companion 버전을 선택할 수 있습니다.

이전 CompanionPi 버전에서 업그레이드

설치 스크립트나 **2.2.0 이상** 이미지로 CompanionPi를 설치했다면 업데이트 도구로 최신 버전까지 쉽게 업데이트할 수 있습니다.

2.2.0 이전 CompanionPi 이미지로 설치했다면 설정을 백업하고 새로 설치한 환경에 다시 가져오는 것을 권장합니다.

참고 2.2.0에서 달라진 점

2.2.0 빌드에서는 CompanionPi 이미지 개편을 비롯해 많은 부분이 바뀌었습니다. 가장 큰 변화는 CompanionPi가 여러 의존성을 직접 관리하게 된 것입니다. 덕분에 이후 버전으로 원활하게 업데이트할 수 있습니다.

1.9 실행 전 준비

Companion을 실행할 컴퓨터

가장 간단한 시작 방법은 주로 사용하는 컴퓨터에서 Companion을 실행하는 것입니다. 지금 이 문서를 읽고 있는 컴퓨터도 괜찮습니다. 다만 항상 켜 두는 별도 컴퓨터나 Raspberry Pi에서 실행하면 더 편리할 때도 있습니다.
같은 네트워크에 연결되어 있다면 어느 쪽이든 사용할 수 있습니다.

Companion의 관리 화면은 네트워크의 다른 컴퓨터에서도 접속할 수 있도록 설정할 수 있습니다. Companion Satellite를 사용하면 Stream Deck과 유사한 제어 장치(Surface)를 네트워크를 통해 Companion에 연결할 수 있습니다.

어떤 컴퓨터를 사용할지 정하기 어렵다면 주로 사용하는 컴퓨터에서 시작하세요. 나중에 옮기더라도 설정을 내보낸 뒤 다시 가져오면 됩니다.

준비 사항

Elgato Stream Deck 앱을 종료합니다. 두 프로그램을 함께 실행할 수도 있지만, Companion이 Stream Deck을 직접 제어할 때 가장 원활하게 작동합니다. 함께 실행해야 한다면 Elgato 스토어의 Companion 플러그인을 사용할 수 있습니다.

Stream Deck이나 지원되는 다른 제어 장치를 컴퓨터에 연결합니다. 제어하려는 장치의 전원을 켜고 네트워크에 연결합니다. 네트워크가 올바르게 설정되어 있고 컴퓨터에서 해당 장치와 통신할 수 있는지 확인합니다.

장치는 나중에도 쉽게 추가할 수 있습니다. 지금 모든 장치가 준비되어 있지 않아도 괜찮습니다.
시작하는 데 필요한 장치만 준비하면 됩니다.

사용 팁

- Companion 관리 화면을 인터넷에 공개하지 마세요. 접속 주소를 발견한 사람이 컴퓨터의 셸을 원격으로 사용할 수 있게 됩니다.
- Companion과 다른 소프트웨어를 같은 컴퓨터에서 실행할 수 있습니다. 소규모 구성에서는 vMix와 Companion을 함께 실행하는 경우가 많고, 규모가 커지면 네트워크를 통해 더 많은 장치나 소프트웨어에 연결합니다.
- 가능하면 Companion에는 항상 유선 네트워크를 사용하세요. WiFi에서 간헐적으로 발생하는 잦은 연결 끊김을 일부 장치의 네트워크 구현이 제대로 처리하지 못할 수 있습니다.
- 설정을 자주 백업하세요. OneDrive/iCloud와 동기화되는 폴더에 매일 자동 백업하도록 쉽게 설정할 수 있습니다. 컴퓨터에 문제가 생겨도 설정 백업으로 빠르게 복구할 수 있습니다.
- 제어하는 장치에는 고정 IP나 DHCP 예약을 사용하여 주소가 예기치 않게 바뀌지 않도록 하세요.
- 장치를 검색할 때 네트워크에서 mDNS/Bonjour 통신이 허용되어 있어야 합니다. 이를 허용하면 여러 장치를 더 쉽게 설정할 수 있습니다.

1.10 서버 시작

Companion을 열면 실행 창이 표시됩니다. 첫 화면에서 네트워크 인터페이스를 선택하고 필요하면 포트를 변경합니다. 기본값은 8000이며, Companion 서버가 사용할 포트입니다.

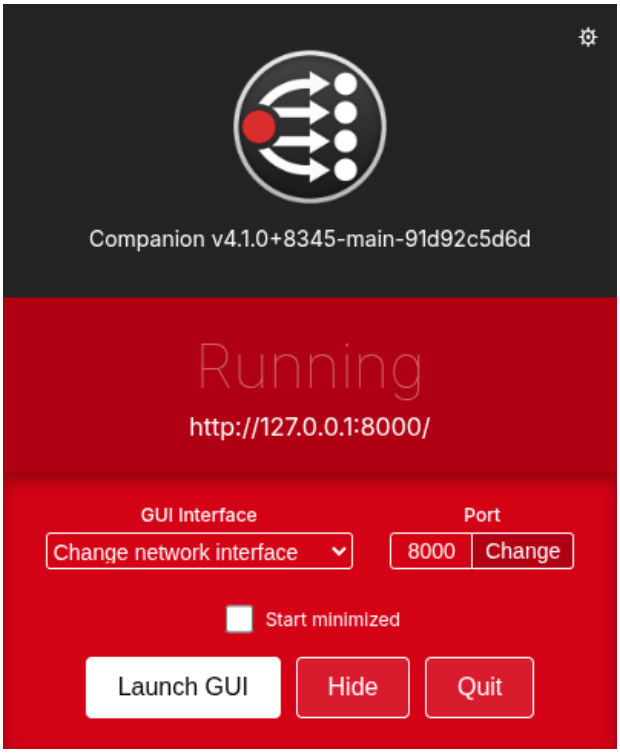


그림 1-2. 실행 창

같은 네트워크의 다른 컴퓨터에서 Companion을 원격으로 제어하려면 'GUI Interface'를 변경하여 다른 네트워크 인터페이스에서도 접속할 수 있게 합니다. 접속 주소는 "Running" 아래에 표시됩니다. Companion이 실행 중인 컴퓨터에서 설정하려면 **관리 화면 열기(Launch GUI)** 버튼을 클릭합니다. 기본 브라우저에서 관리 페이지가 열립니다.

Google Chrome 사용을 권장하지만 최신 상태로 업데이트된 다른 브라우저도 사용할 수 있습니다.
구형 Android 및 iOS 장치의 내장 브라우저에는 알려진 문제가 있습니다.

실행 창에는 **최소화 상태로 시작(Start minimized)** 옵션이 있습니다. macOS와 Windows에서는 로그인할 때 Companion을 자동으로 시작하는 **로그인 시 실행(Run at login)** 옵션도 제공됩니다.

설정 창

오른쪽 위의 톱니바퀴를 클릭하면 실행 프로그램의 설정 창이 열립니다. 서버 실행과 관련된 상세 옵션이 다음 구역으로 나뉘어 있습니다.

- **일반(General)** - 로그 수준을 설정합니다.
- **위험 기능(Dangerous Features)** - **셸 명령 지원(Shell command support)**과 **제한된 모듈(Restricted modules)**의 사용 여부를 설정합니다. 두 옵션은 기본적으로 꺼져 있습니다. 역방향 프록시용 **신뢰할 프록시(Trusted proxies)**도 설정할 수 있습니다.
- **개발자(Developer)** - 개발자 모듈을 활성화하고 로컬 모듈 폴더를 지정합니다. 모듈 개발자가 사용하는 옵션이며, 자세한 내용은 모듈 개발 문서를 참조하세요.
- **Syslog** - Companion 로그를 syslog 서버로 전달합니다.

각 옵션의 기능은 서버 구성에 설명되어 있습니다. 보안 옵션을 변경하면 Companion이 다시 시작됩니다.

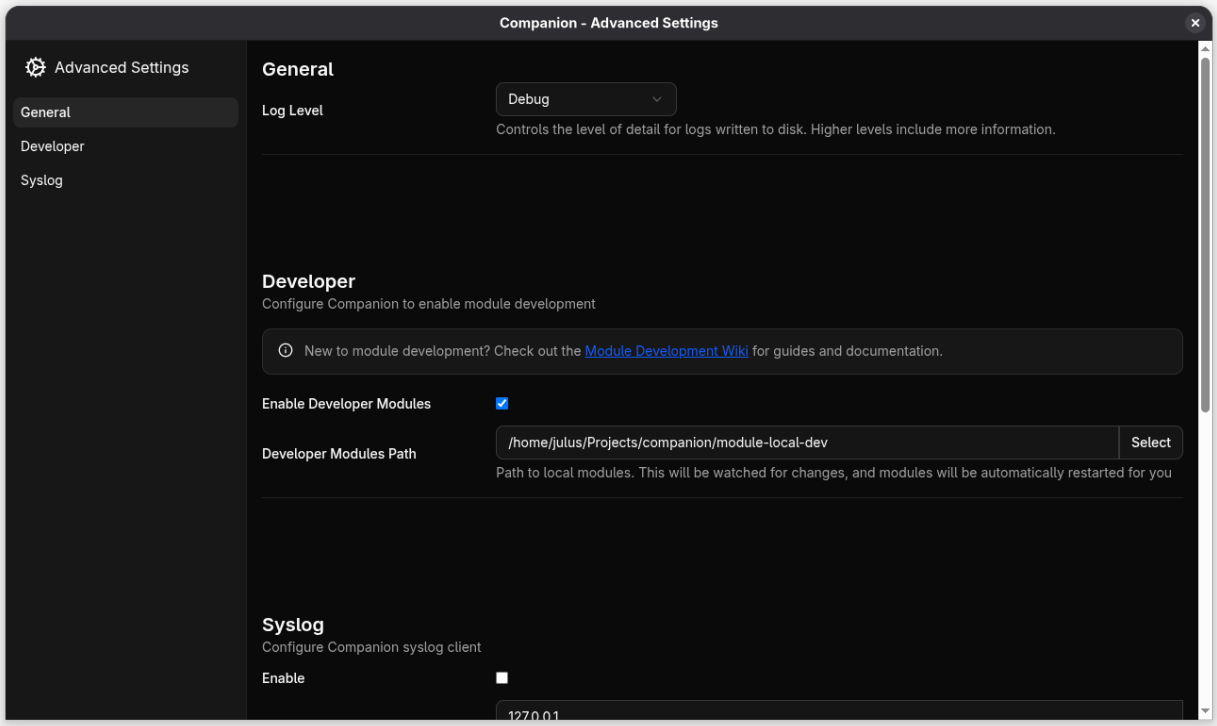


그림 1-3. 실행 프로그램 고급 설정

1.11 서버 구성

도움말 CompanionPi나 화면 없는 서버에서 실행하나요?

이 페이지에서는 옵션별 기능을 설명합니다. 실제 설정은 Headless 구성에 따라 명령줄에서 변경해야 합니다.

이 설정은 **Companion 서버 자체의 시작 방식**을 제어합니다. 수신할 네트워크 포트와 허용할 보안 기능 등이 여기에 해당합니다. 이미 실행 중인 Companion의 동작을 설정하는 관리 화면의 설정(Settings) 페이지와는 별개입니다.

실행 환경에 따라 다음 위치에서 설정합니다.

- **데스크톱 설치**: 실행 프로그램의 설정 창을 사용합니다. 서버 시작을 참조하세요.
- **Headless / 서버 설치**: config-tool을 사용합니다. Headless 구성을 참조하세요.

두 방식 모두 아래에서 설명하는 동일한 옵션을 편집합니다. 일부 옵션, 특히 보안 옵션을 변경하면 새 설정을 적용하기 위해 Companion이 다시 시작됩니다.

정확한 config.yaml 키, 환경 변수, 명령줄 플래그를 포함한 전체 옵션은 설정 항목 참조를 확인하세요.

네트워크

- **관리 포트(Admin port)** - 관리 화면과 나머지 웹 UI가 수신할 포트입니다. 기본값은 **8000**입니다.
- **관리 주소 / 인터페이스(Admin address / interface)** - 관리 화면이 바인딩할 네트워크 주소 또는 인터페이스입니다. 기본적으로 모든 인터페이스에서 수신합니다. 특정 네트워크에서만 Companion에 접속하도록 하려면 이 값을 제한합니다.
- **신뢰할 프록시(Trusted proxies)** - 역방향 프록시 뒤에서 Companion을 실행할 때 설정합니다. 실제 클라이언트 주소를 확인할 수 있도록 신뢰할 프록시 주소를 지정합니다. 설정하지 않으면 원격 클라이언트를 로컬 클라이언트로 잘못 판단할 수 있으며, 이는 아래의 보안 기능에 영향을 줍니다. loopback 또는 쉼표로 구분한 프록시 IP 주소 목록을 입력할 수 있습니다.
- **IPv6 사용 안 함(Disable IPv6)** - IPv6로 인해 문제가 발생하는 네트워크에서 관리 화면의 IPv6 바인딩을 끕니다.

로깅

- **로그 수준(Log level)** - Companion이 로그에 기록할 상세 수준입니다. error, warn, info, debug 등의 수준을 사용할 수 있습니다. 상세 수준을 높이면 문제 진단에 유용한 정보가 더 많이 기록되지만 일상적으로는 로그가 많아집니다.

Syslog

Companion 로그를 syslog 서버로 전달할 수 있습니다. 서버에 배포하여 로그를 중앙에서 관리할 때 유용합니다.

- **Syslog 사용(Enable syslog)** - syslog 전달을 켜거나 끕니다.
- **Syslog 서버(Syslog server)** - syslog 서버 주소입니다. 기본값은 localhost입니다.
- **포트(Port)** - syslog 서버의 포트입니다. syslog 기본 포트는 514입니다.
- **TCP 사용(Use TCP)** - UDP 대신 TCP로 전달합니다.
- **로컬 호스트 이름(Local hostname)** - Companion이 syslog 서버에 전달할 호스트 이름입니다. 지정하지 않으면 시스템 호스트 이름을 사용합니다.

보안(위험 기능)

다음 두 기능은 영향 범위가 크므로 **기본적으로 비활성화**되어 있습니다. 특히 네트워크를 통해 Companion에 접속할 수 있다면 신뢰할 수 있는 환경에서만 활성화해야 합니다. 어느 옵션을 변경하든 Companion이 다시 시작됩니다.

- **셸 명령 지원(Shell command support)** - Companion을 실행하는 컴퓨터에서 셸 명령을 실행하도록 허용합니다. 내부 액션인 **셸 명령 실행(Run shell command)** 등이 이 기능을 사용합니다. 기본적으로 꺼져 있습니다.
- **제한된 모듈(Restricted modules)** - 안전상의 이유로 제한된 모듈을 불러오도록 허용합니다. 원격 클라이언트가 보낸 사용자 지정 모듈 가져오기 등이 해당합니다. Companion이 실행 중인 컴퓨터에서 모듈을 가져오는 것은 항상 허용되며, 이 설정은 원격 클라이언트에만 적용됩니다. 데스크톱 설치에서는 기본적으로 꺼져 있습니다. Headless 설치에서는 보통 다른 컴퓨터에서 모듈을 가져오므로 config-tool이 기본적으로 켭니다.

주의

이 기능은 Companion이 실행되는 컴퓨터에서 코드를 실행할 수 있게 합니다. 위험을 이해하고
감수할 수 있을 때만 활성화하세요.

경로 및 개발자 옵션

- **추가 모듈 경로 / 개발자 모듈(Extra module path / developer modules)** - 설치된 모듈 외에 불러올 로컬 모듈 폴더를 지정합니다. 주로 모듈 개발자가 사용합니다. 폴더의 변경 사항을 감시하며 모듈이 바뀌면 자동으로 다시 시작합니다. 모듈 개발 문서를 참조하세요.

고급

- **머리글 알림(Header notifications)** - 관리 화면 상단에 버전 관련 알림을 표시할지 설정합니다. 업데이트 안내를 보고 싶지 않다면 끌 수 있습니다.
- **설치 이름(Installation name)** - 이 Companion 설치에 고정 이름을 지정합니다. 설정하면 설정의 **Installation Name**을 대체하며, 해당 화면에서는 더 이상 이름을 변경할 수 없습니다. 데이터베이스를 초기화해도 이 이름은 유지됩니다. 스크립트나 이미지로 여러 Companion을 배포할 때 유용합니다. 데스크톱 설치에서는 COMPANION_INSTALL_NAME 환경 변수로 지정할 수 있습니다.

1.12 설정 항목 참조

Companion 서버의 시작 방식을 제어하는 모든 실행 옵션입니다.

Headless 또는 Docker 설치에서는 아래의 **키(Key)**에 해당하는 이름으로 config.yaml에 설정합니다.
파일 위치는 Docker 이미지에서 /companion/config.yaml, CompanionPi에서 /etc/companion/config.yaml입니다.
CompanionPi에서는 config-tool이 파일을 편집해 줍니다. Docker 이미지에서는 파일을 직접 편집하면 시작할 때
진입점 스크립트가 적용합니다. 서버를 직접 실행할 때는 같은 옵션을 **명령줄 플래그** 또는 **환경 변수**로 전달할 수 있습니다.

옵션별 _기능_과 변경이 필요한 상황은 서버 구성을 참조하세요.

네트워크

adminPort

number · 기본값 8000 · 명령줄 --admin-port

웹 관리 화면과 API가 수신할 TCP 포트입니다. 기본값은 8000입니다.

adminInterface

string · 명령줄 --admin-interface

지정한 이름의 네트워크 인터페이스(예: eth0)에서 첫 번째 IPv4 주소에 관리 화면을 바인딩합니다.
인터페이스 대신 바인딩 주소를 사용하려면 지정하지 않습니다.

adminAddress

string · 명령줄 --admin-address

관리 화면을 바인딩할 IP 주소입니다. 지정하지 않으면 모든 인터페이스(0.0.0.0 / ::)에 바인딩합니다.
바인딩 인터페이스와 동시에 지정할 수 없습니다.

trustedProxies

string · 명령줄 --trusted-proxies · 환경 변수 COMPANION_TRUSTED_PROXIES

역방향 프록시 뒤에서 Companion을 실행할 때 프록시 주소 대신 실제 방문자의 IP 주소를 감지하도록 사용합니다.
'loopback' 또는 신뢰할 프록시 IP 주소나 서브넷을 쉼표·세미콜론으로 구분한 목록을 설정합니다.

disableIpv6

boolean · 기본값 false · 환경 변수 DISABLE_IPV6

IPv6 지원을 끕니다. 활성화하면 Companion의 여러 API 및 웹 화면 포트가 기본적으로 :: 대신 IPv4(0.0.0.0)에 바인딩합니다.

로깅

logLevel

error, warn, info, http, verbose, debug, silly 중 하나 · 명령줄 --log-level

콘솔 로그의 상세 수준입니다. error, warn, info, http, verbose, debug, silly 중 하나를 사용합니다.
지정하지 않으면 기본값(info)을 사용합니다.

Syslog

syslogEnable

boolean · 기본값 false · 명령줄 --syslog-enable

로그를 syslog 서버로 전달합니다. 나머지 syslog 옵션은 이 옵션이 켜져 있을 때만 적용됩니다.

syslogHost

string · 명령줄 --syslog-host

로그를 기록할 syslog 서버의 호스트 이름 또는 IP 주소입니다. 기본값은 localhost입니다.

syslogPort

number · 명령줄 --syslog-port

로그를 기록할 syslog 서버의 포트입니다. 기본값은 514입니다.

syslogTcp

boolean · 기본값 false · 명령줄 --syslog-tcp

syslog 전송에 기본 UDP 대신 TCP를 사용합니다.

syslogLocalhost

string · 명령줄 --syslog-localhost

이 컴퓨터가 syslog 서버에 전달할 호스트 이름입니다. 지정하지 않으면 시스템 호스트 이름을 사용합니다.

보안

enableShellCommandSupport

boolean · 기본값 false · 명령줄 --enable-shell-command-support ·
환경 변수 COMPANION_ENABLE_SHELL_COMMAND_SUPPORT

이 컴퓨터에서 셸 명령을 실행하도록 허용합니다. 내부 액션인 "run shell command" 등이 해당하며,
기본적으로 꺼져 있습니다.

enableRestrictedModules

boolean · 기본값 true · 명령줄 --enable-restricted-modules ·
환경 변수 COMPANION_ENABLE_RESTRICTED_MODULES

안전상의 이유로 제한된 모듈을 불러오도록 허용합니다. 원격(루프백이 아닌) 클라이언트에서 사용자 지정 모듈을 가져오는 경우 등이 해당합니다. 기본적으로 켜져 있습니다.

경로

extraModulePath

string · 명령줄 --extra-module-path

불러올 개발 중 모듈을 추가로 검색할 디렉터리입니다.

고급

notifications

boolean · 기본값 true · 명령줄 --no-notifications

관리 화면 상단에 버전 관련 알림을 표시합니다. 기본적으로 켜져 있습니다.

installName

string · 명령줄 --install-name · 환경 변수 COMPANION_INSTALL_NAME

이 Companion 설치에 사용할 고정 이름입니다. 지정하면 관리 화면의 값을 대체하고 잠급니다. 관리 화면에서 변경할 수 없으며 데이터베이스를 초기화해도 유지됩니다. 화면에서 이름을 관리하려면 지정하지 않습니다.

서버 전용 옵션

이 옵션은 config.yaml 대신 실행 프로그램이나 컨테이너 진입점에서 설정하므로 명령줄 플래그로만 사용할 수 있습니다.

- --config-dir <path> - 설정과 데이터를 저장할 디렉터리입니다. 기본값은 플랫폼마다 다릅니다.
- --list-interfaces - 관리 화면의 바인딩 인터페이스로 사용할 수 있는 네트워크 인터페이스 목록을 출력한 뒤 종료합니다.

1.13 고급 설정

도움말

이 페이지에서는 드물게 필요한 고급 설정을 설명합니다. 필요한지 확실하지 않다면 이 페이지를 건너뛰어도 괜찮습니다.

역방향 프록시 설정

드물게 역방향 프록시를 통해 Companion을 제공해야 하는 경우가 있습니다.

다른 역방향 프록시의 설정 예제가 있다면 GitHub 이슈로 알려주세요. 여기에 추가할 수 있습니다.

일반적으로 WebSocket 처리만 확인하면 특별한 설정은 필요하지 않습니다. nginx의 전체 설정 예제는 다음과 같습니다.

```
  location / {
    proxy_pass http://127.0.0.1:8000/;

    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;

    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";

    resolver 127.0.0.11;
  }
```

하위 경로 사용

도메인의 최상위 경로 대신 하위 경로에서 제공해야 할 수도 있습니다.
Companion 4.1부터 지원합니다. 널리 사용되는 방식은 아니므로 새 기능에서 이 사용 방식을 간혹 고려하지 못할 수 있습니다. 이런 문제를 발견하면 이슈를 등록해 주세요. 수정하겠습니다.

```
location /my-subpath-here/ {
    proxy_set_header Companion-custom-prefix "my-subpath-here";
    proxy_pass http://127.0.0.1:8000/;

    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;

    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";

    resolver 127.0.0.11;
}
```

2 설정 화면 사용

2.1 사용자 화면

장치와 제어 장치 설정, 버튼·변수·트리거 정의, 기타 작업과 진단 등 모든 페이지에서 공통으로 사용하는 화면 구성을 설명합니다.

기본 화면 구성

- 왼쪽 사이드바에서 각 페이지로 이동합니다.
- 페이지 상단에는 Companion 새 버전 알림 등이 표시되며 도움말과 지원 메뉴에 빠르게 접근할 수 있습니다.
  관리자 비밀번호를 사용하면 상단에 자물쇠 아이콘이 나타나 한 번의 클릭으로 관리 화면을 잠글 수 있습니다.
- 페이지 본문은 일반적으로 주요 작업을 위한 큰 기본 패널(왼쪽)과 상세 정보·미리보기·관련 설정을 위한 보조 패널(오른쪽)로 나뉩니다.
- 넓은 화면에서는 두 패널 사이의 구분선을 드래그하여 어느 한쪽을 더 넓힐 수 있습니다. 구분선을 두 번 클릭하면 가운데로 돌아갑니다. 각 페이지의 구분선 위치는 브라우저에 개별 저장됩니다.
- 작은 화면에서는 추가 탐색 메뉴를 통해 기본 패널과 보조 패널을 함께 제공합니다. 스크롤을 줄이면서 관련 설정에 쉽게 접근할 수 있도록 구성됩니다.

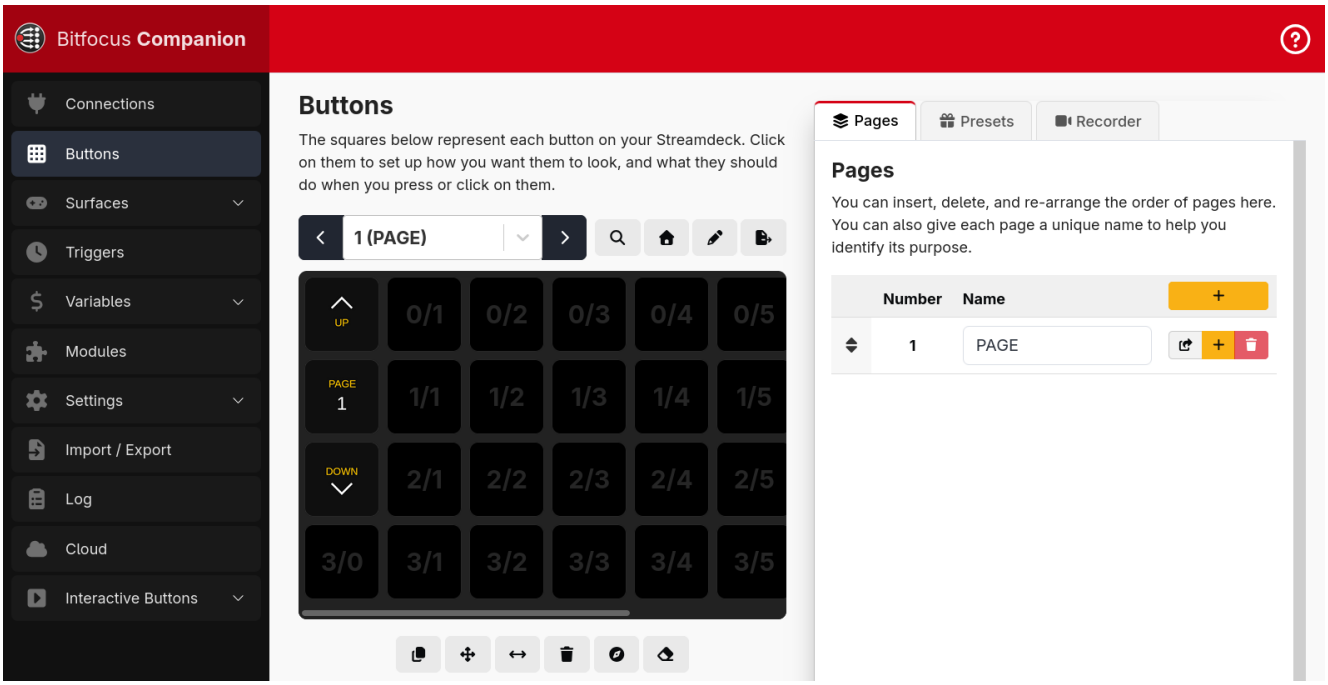


그림 2-1. 관리 화면

사이드바 옵션

4.3부터 새 컨텍스트 메뉴로 사이드바 사용 방식을 변경할 수 있습니다. 일반적으로 마우스 오른쪽 버튼을 클릭하면 메뉴가 열립니다.

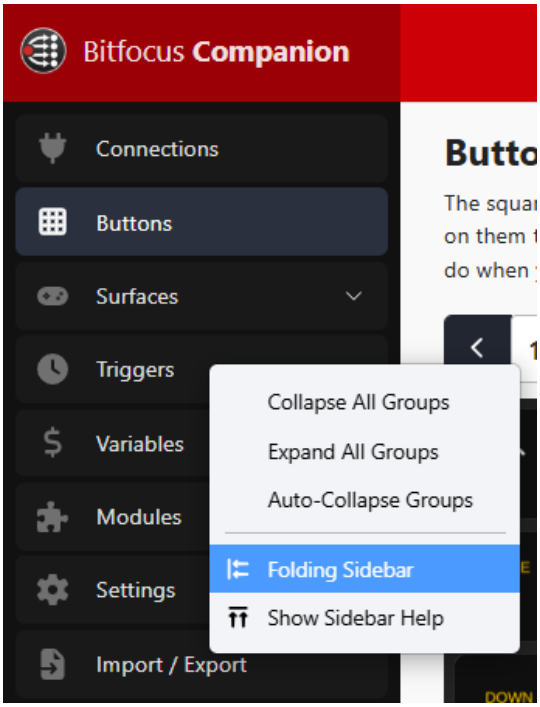


그림 2-2. 사이드바 컨텍스트 메뉴

- **모든 그룹 접기(Collapse All Groups)/모든 그룹 펼치기(Expand All Groups)**는 모든 최상위 그룹을 접거나 펼칩니다. Surfaces와 Variables처럼 오른쪽에 꺾쇠 표시가 있는 항목이 해당합니다.
- **그룹 자동 접기(Auto-Collapse Groups)**를 사용하면 한 번에 하나의 최상위 그룹만 열립니다. 한 그룹을 열면 나머지는 모두 닫힙니다. 아코디언 모드라고도 합니다. 활성화하면 컨텍스트 메뉴의 해당 항목 옆에 체크 표시가 나타납니다.
- **접이식 사이드바(Folding Sidebar)/전체 너비 사이드바(Full-width Sidebar)**는 두 표시 방식을 전환합니다. 접이식에서는 아이콘만 표시되다가 마우스를 올리면 펼쳐집니다. 사이드바 왼쪽 아래의 아이콘으로도 전환할 수 있습니다. 창이 매우 좁은 모바일 모드에서는 이 옵션을 사용할 수 없습니다.
- **사이드바 접힌 상태 유지(Keep Sidebar Folded)**를 사용하면 마우스를 올려도 좁은 상태로 유지됩니다.
- **모듈 변수 숨기기(Hide Module Variables)/모듈 변수 표시(Show Module Variables)**는 사이드바의 Variables 아래에 각 연결을 표시할지 전환합니다. 숨겨도 기본 Variables 페이지에서는 항상 확인할 수 있습니다.
- **사이드바 도움말 표시/숨기기(Show/Hide Sidebar Help)**는 사이드바 아래쪽의 스크롤되지 않는 영역에 있는 도움말과 지원 항목을 표시하거나 숨깁니다. 같은 항목이 창 오른쪽 위의 도움말 메뉴에도 있으므로 여기에서 숨기면 스크롤하지 않고 더 많은 탐색 버튼을 볼 수 있습니다.

도움말 메뉴

4.3부터 다른 브라우저 기반 앱처럼 창 오른쪽 위, 빨간색 상단 표시줄의 오른쪽 끝에 있는 메뉴에서 도움말을 열 수 있습니다. 사이드바 아래쪽과 동일한 항목을 제공합니다. 중복 도움말 항목을 숨겨 사이드바 공간을 확보하는 방법은 위의 사이드바 옵션을 참조하세요.

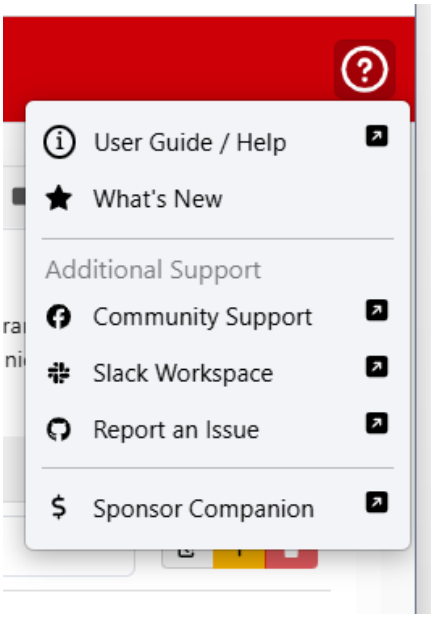


그림 2-3. 상단 도움말 메뉴

목록과 트리

프리셋, 변수, 추가할 수 있는 액션과 피드백 등 Companion의 여러 목록은 펼치거나 접을 수 있는 구역으로 나뉩니다. **Alt**를 누른 채 구역을 클릭하면 같은 수준의 구역을 한 번에 모두 펼치거나 접을 수 있습니다.

2.2 연결

연결(Connections) 탭에서 장치를 추가·설정·제거할 수 있습니다. 표에는 각 연결(Connection)의 상태와 관련 정보도 표시됩니다.

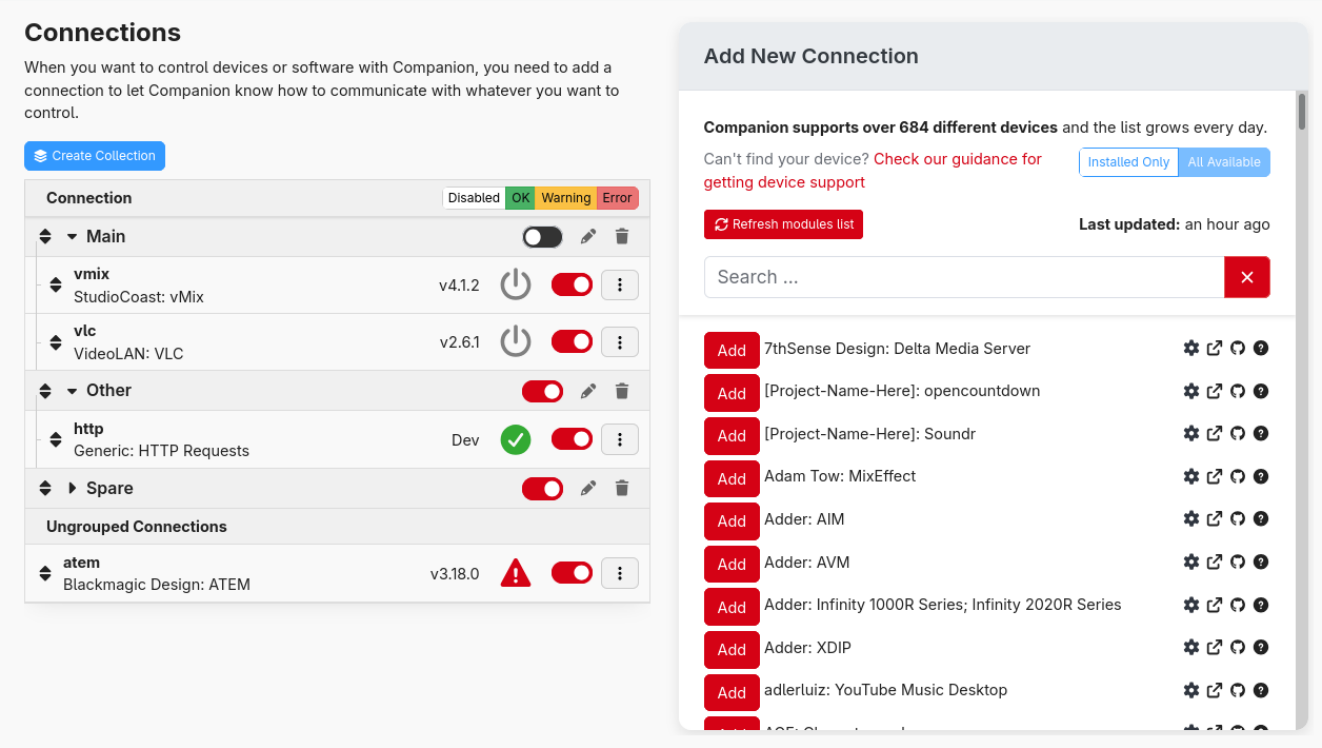


그림 2-4. 연결 페이지

4.0부터 모듈은 다운로드할 수 있는 플러그인으로 제공됩니다. 이전보다 모듈 업데이트를 더 빠르고 쉽게 받을 수 있으며, 이에 따라 Companion에는 모듈이 함께 포함되지 않습니다.

참고

Companion을 업데이트해도 모듈은 더 이상 자동 업데이트되지 않습니다. 준비가 되었을 때 이 페이지에서 직접 업데이트해야 합니다.

설정을 가져올 때도 이 페이지에서 적절한 모듈 버전이 선택되고 설치되어 있는지 확인해야 합니다.

오프라인 시스템에서는 오른쪽 목록이 비어 있을 수 있습니다. 웹사이트에서 오프라인 모듈 번들을 다운로드하여 가져와야 합니다. 자세한 내용은 모듈을 참조하세요.

연결 추가

필요한 모듈 버전을 아직 설치하지 않았다면 목록을 전환하여 설치된 모듈뿐 아니라 설치 가능한 모듈도 표시해야 할 수 있습니다.

1. 목록을 스크롤하거나 검색하여 추가할 새 장치를 찾습니다.
2. 추가하려는 장치를 선택합니다.
3. 대화 상자에서 사용할 모듈 버전을 선택합니다. 설치된 버전은 언제든 선택할 수 있으며, 스토어에 접속할 수 있으면 최신 버전 설치도 제안됩니다.

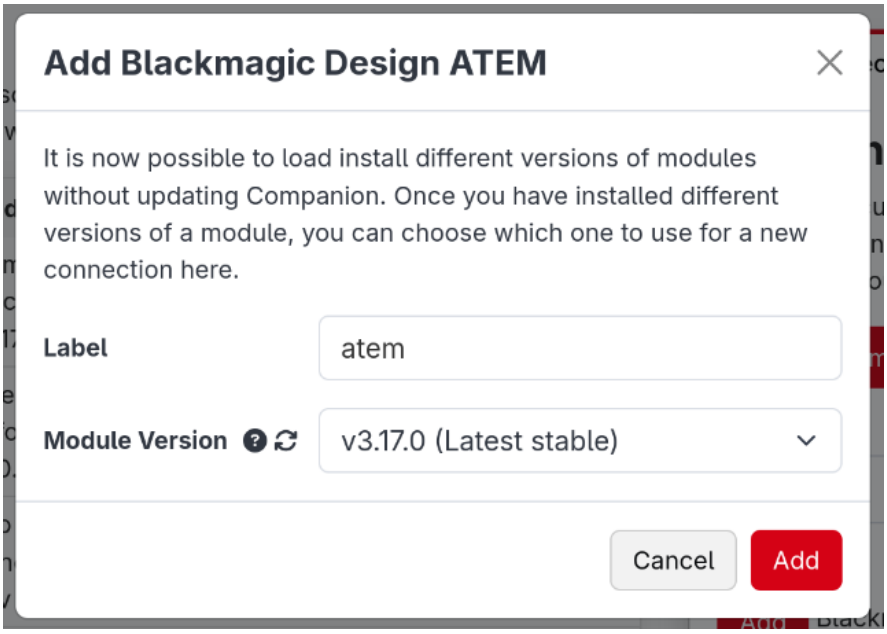


그림 2-5. 연결 추가 대화 상자

4. 장치의 연결 정보를 입력하고 변경 사항을 저장합니다.

추가한 장치가 제어할 다른 장치들과 함께 Connections 탭에 표시됩니다. 각 장치는 별도의 연결이 필요합니다. Barco E2가 두 대라면 각각 다른 연결로 추가해야 합니다.

참고

같은 모듈을 사용하는 여러 연결에서도 연결마다 다른 모듈 버전을 실행할 수 있습니다.

연결을 추가한 뒤 표에서 순서를 바꾸거나 필터링할 수 있습니다. 맨 오른쪽 열에서는 다음 버튼 목록을 펼칠 수 있습니다.

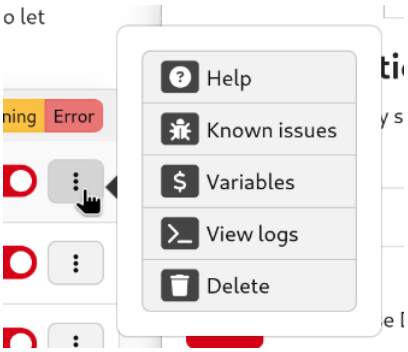


그림 2-6. 연결 팝업 메뉴

- **도움말(Help)**은 모듈의 도움말을 엽니다. 연결을 추가하기 전에도 오른쪽 목록에서 열 수 있습니다.

- **알려진 문제(Known Issues)**는 해당 모듈의 GitHub 페이지로 이동합니다. 모듈 사용 중 발견한 오류나 필요한 기능을 이곳에 등록하세요. 오류나 기능 요청을 구체적으로 설명할수록 더 빨리 처리될 가능성이 높습니다.
- **변수(Variables)**는 해당 연결에서 제공하는 모든 변수를 표시합니다. 변수는 뒤에서 설명하는 여러 위치에서 사용할 수 있습니다.
- **로그 보기(View Logs)**는 모듈의 디버그 로그를 표시합니다. 오류를 신고하면 모듈 개발자가 원인 파악을 위해 이곳의 추가 정보를 요청할 수 있습니다. 대부분의 사용자는 평소에 확인할 필요가 없습니다.

지원 장치 전체 목록은 웹사이트의 Companion 모듈 지원 목록에서도 확인할 수 있습니다.

컬렉션

연결이 많다면 컬렉션으로 정리할 수 있습니다. 연결 표 위의 **컬렉션 만들기(Create Collection)**를 클릭하고 이름을 지정한 뒤 연결을 안으로 드래그합니다. 컬렉션 안에 다른 컬렉션을 넣을 수도 있습니다.

각 컬렉션에는 자체 사용 스위치가 있습니다. 컬렉션을 끄면 하위 컬렉션을 포함한 모든 연결이 비활성화되므로 여러 연결을 한 번에 끌 때 편리합니다. 개별 연결의 사용 설정은 그대로 유지되며, 컬렉션을 다시 켜면 이전 상태로 돌아갑니다.

연결 모니터링

Companion은 연결 상태를 확인할 수 있는 내부 변수를 제공합니다.

변수	값
$(internal:instance_total)	연결 수
$(internal:instance_oks)	정상인 연결 수
$(internal:instance_warns)	경고가 있는 연결 수
$(internal:instance_errors)	오류가 있는 연결 수
$(internal:instance_disabled)	비활성화된 연결 수
$(internal:connection_<label>_status)	연결 상태: good, warning, error, disabled 중 하나
$(internal:connection_collection_<id>_status)	연결 컬렉션의 활성화(true) 또는 비활성화(false) 여부

컬렉션 상태 변수에는 컬렉션 이름 대신 내부 ID를 사용합니다. 해당 변수를 찾는 가장 쉬운 방법은 변수 페이지의 **내부(Internal)** 구역을 확인하는 것입니다. 각 변수의 설명은 **Connection Collection Status:** 다음에 컬렉션 이름이 붙는 형태입니다.

버튼에 연결 상태를 표시하는 내부 피드백도 있습니다.

연결 구성

연결을 추가한 뒤 목록에서 해당 연결을 클릭하면 오른쪽 패널에 설정이 열립니다.

연결이 활성화되어 있으면 모듈에서 정의한 설정을 변경할 수 있습니다.

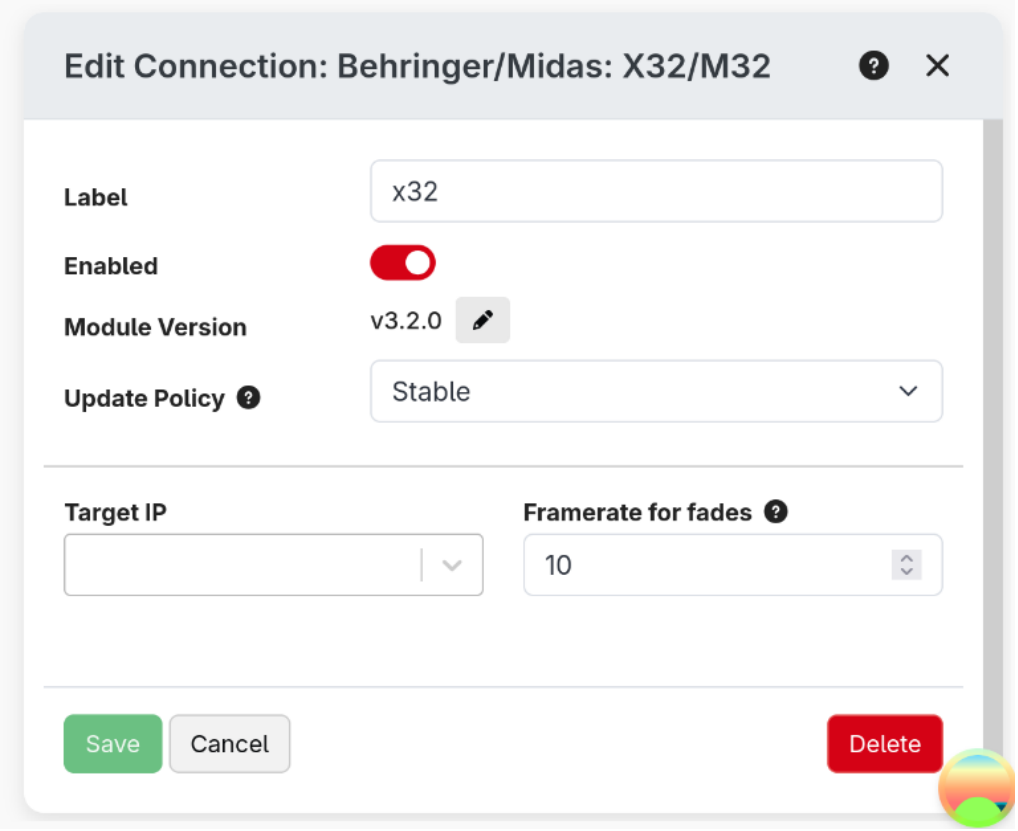


그림 2-7. 연결 설정 패널

변경을 마쳤으면 반드시 저장(Save)을 클릭하여 저장합니다.

연결 업데이트

연결 설정 패널에서 사용 중인 모듈의 버전도 업데이트할 수 있습니다.

모듈 버전 번호 옆의 연필 아이콘을 클릭하여 편집기를 엽니다.

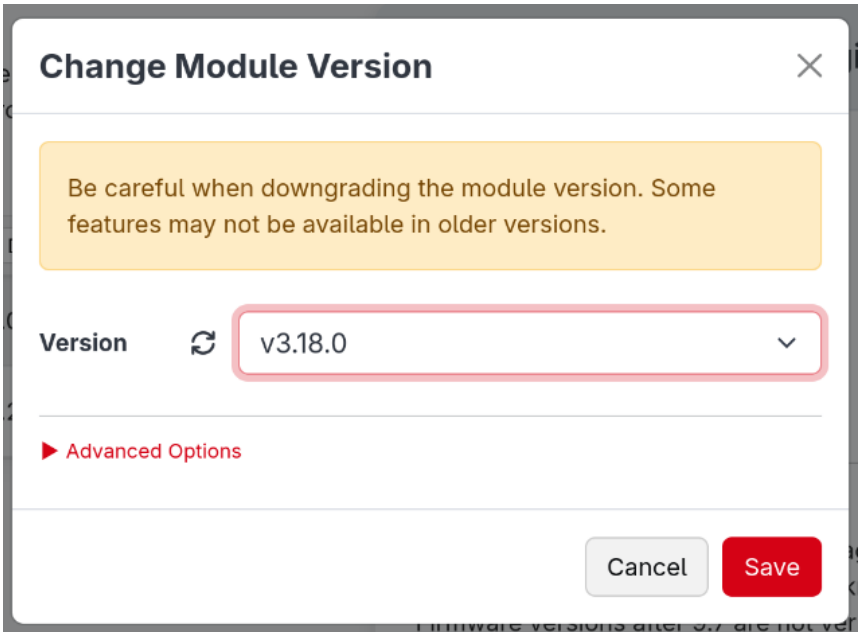


그림 2-8. 연결의 모듈 버전 업데이트

이 패널에는 최신 사용 가능 버전과 설치된 버전이 표시됩니다. Save를 클릭하면 연결을 다시 시작하여 변경 사항을 적용합니다.

도움말

모듈 페이지에서 다른 모듈 버전을 추가로 설치하거나 설치된 버전을 제거할 수 있습니다.

사용이 권장되지 않는 모듈

모듈 관리자에 의해 모듈 전체나 특정 버전이 비권장(deprecated)으로 지정될 수 있습니다. 일반적으로 더 이상 업데이트하지 않거나, 새 버전 또는 다른 모듈을 대신 사용해야 한다는 뜻입니다.

이 상태는 다음과 같이 명확히 표시됩니다.

- 연결 목록의 모듈 이름 옆에 노란색 **Deprecated** 또는 **Deprecated version** 표시가 나타납니다.
  마우스를 올리면 이유를 확인할 수 있습니다.
- 연결 설정 패널 위쪽과 모듈 도움말에 경고가 표시됩니다.

특정 버전만 해당한다면 앞에서 설명한 방법으로 연결을 새 모듈 버전으로 업데이트합니다. 모듈 전체가 해당한다면 장치에 사용할 수 있는 대체 모듈을 찾습니다.

이 정보는 모듈 스토어에서 가져오므로 Companion이 스토어에 접속한 경우에만 표시됩니다.

필요하다면 연결이 다른 모듈을 사용하도록 강제로 변경할 수 있습니다. 일반적인 사용에는 필요하지 않으며,
개발자가 예상하지 못한 예외 상황을 해결하기 위한 방법입니다.

2.3 버튼

버튼(Buttons) 탭에서 Stream Deck에 사용할 버튼을 설정합니다.

도움말

버튼 설정 페이지에서 그리드 크기를 원하는 대로 바꿀 수 있습니다. 4방향 모두로 확장할 수 있습니다.

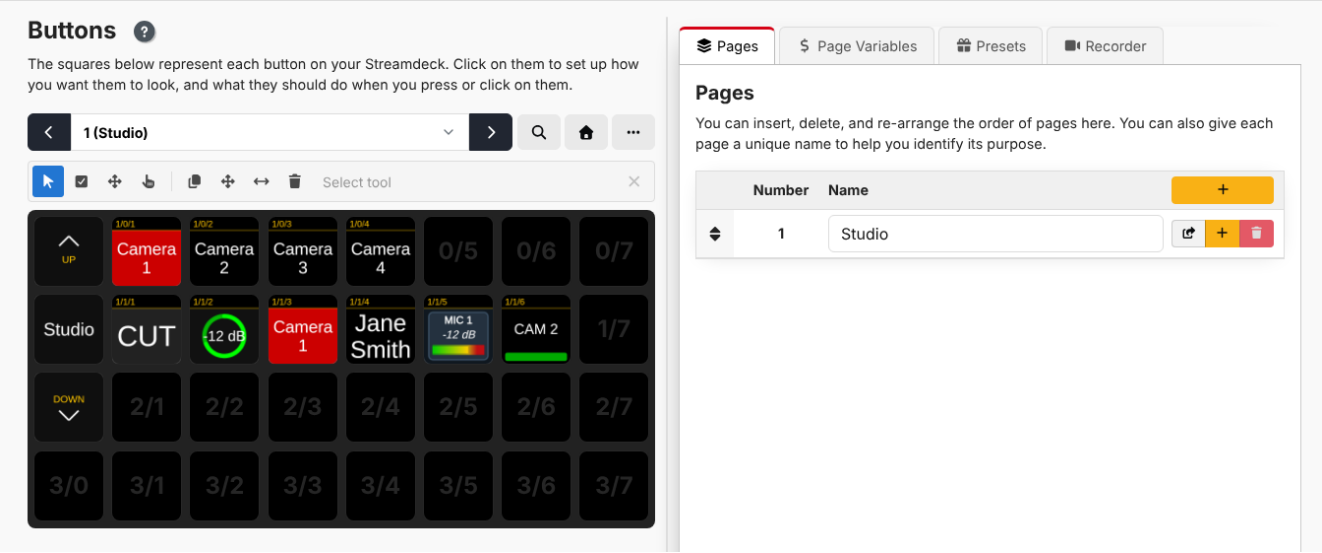


그림 2-9. 버튼 페이지

버튼 배치는 여러 페이지로 구성되며, 좌우 화살표나 드롭다운으로 페이지를 이동할 수 있습니다. 오른쪽 패널의 **페이지(Pages)** 탭에서도 페이지 목록을 보고 이름을 지정하거나 관리할 수 있습니다. 자세한 내용은 페이지를 참고하십시오.

페이지 선택기 옆에는 다음 기능이 있습니다.

- **보기 배율(View Scale)** - 돋보기 아이콘으로, 그리드의 표시 크기를 조절합니다.
- **처음 위치(Home Position)** - 큰 그리드를 스크롤한 뒤 화면을 0/0(왼쪽 위) 위치로 되돌립니다.
- **페이지 작업(Page actions)** (...) - 페이지 전체에 적용할 작업 메뉴입니다.
  - **페이지 편집(Edit page)** - 현재 페이지의 설정을 변경합니다.
  - **페이지 내보내기(Export page)** - 현재 페이지의 버튼을 내려받아 다른 페이지나 Companion 설정으로 가져올 수 있습니다. 가져오기/내보내기를 참고하십시오.
  - **이동 버튼 다시 만들기(Recreate navigation buttons)** - 첫 번째 열의 기존 내용을 기본 페이지 위로, 페이지 번호, 페이지 아래로 버튼으로 대체합니다.
  - **페이지 비우기(Clear page)** - 페이지의 모든 버튼을 지웁니다.

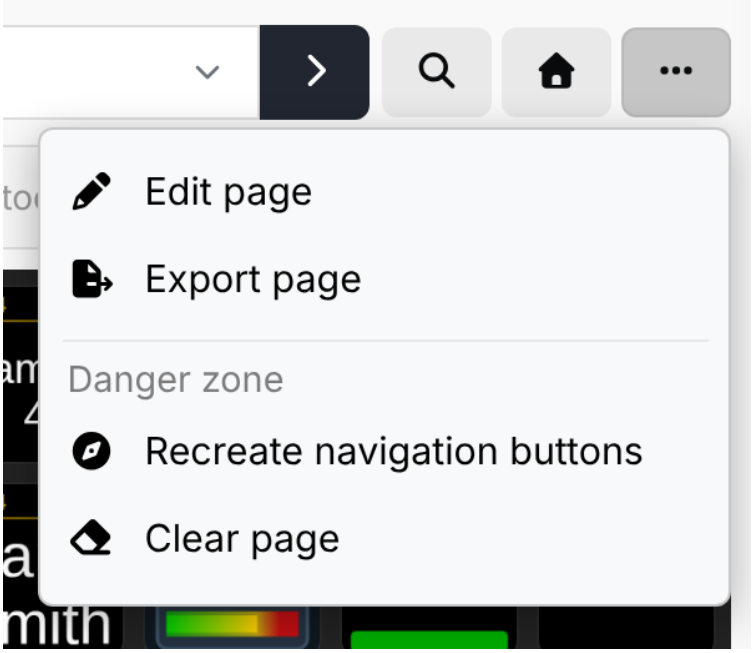


그림 2-10. 페이지 작업 메뉴

버튼 선택과 배치

그리드 위의 도구 모음에서 그리드를 클릭할 때 수행할 작업을 선택합니다. 도구 위에 마우스를 올리면 이름을 볼 수 있습니다.



그림 2-11. 그리드 도구 모음

첫 번째 도구 그룹은 버튼 선택 방식을 정합니다.

- **선택(Select)** - 기본 도구입니다. 버튼을 클릭하면 선택되고 편집기가 열립니다. 그리드에서 사각형을 드래그해 여러 버튼을 한꺼번에 선택하거나, **Shift**를 누른 채 클릭해 범위를 선택할 수도 있습니다. **Ctrl/Cmd**를 누른 채 클릭하면 버튼 하나를 선택에 추가하거나 선택에서 제외합니다.
- **다중 선택(Multi)** - 클릭할 때마다 해당 버튼을 선택에 추가하거나 선택에서 제외합니다. 터치스크린에서 여러 버튼을 선택하기에 가장 편리한 방법입니다.

- **배치(Arrange)** - 편집기를 열지 않고 버튼을 드래그해 이동합니다.
- **누르기(Press)** - 클릭한 버튼이 실제로 눌려 액션이 실행됩니다. 제어 장치에서 버튼을 누르는 것과 같습니다.

두 번째 도구 그룹은 버튼을 재배치합니다. 도구를 고르고 작업할 버튼을 클릭한 뒤 이동할 위치를 클릭합니다.
버튼을 이미 선택했다면 다시 선택할 필요가 없습니다. 도구 모음 끝의 안내 문구가 다음에 해야 할 작업을 알려 줍니다.

- **복사(Copy)** - 버튼을 다른 위치로 복사합니다. 다른 페이지로도 복사할 수 있습니다.
- **이동(Move)** - 버튼을 다른 위치로 옮깁니다.
- **맞바꾸기(Swap)** - 두 버튼의 위치를 맞바꿉니다.
- **삭제(Delete)** - 버튼을 비웁니다.

도구는 사용한 뒤에도 활성 상태를 유지하므로 같은 버튼을 여러 번 연속해서 복사할 수 있습니다. 도구를 다시 클릭하거나 **Escape**를 누르면 **Select**로 돌아갑니다. 도구 모음 끝의 ×는 진행 중인 그리드 작업을 취소하고 선택을 해제합니다.

여러 버튼을 선택한 상태에서는 전체를 함께 드래그해 옮길 수 있습니다. 이동할 위치에 기존 버튼이 하나라도 있으면 Companion이 덮어쓰기 전에 확인을 요청합니다.

참고

이전 Companion에서는 **Shift**를 누른 채 버튼을 클릭해 누름 동작을 실행할 수 있었습니다.
현재는 **Press** 도구가 이 기능을 대신합니다.

오른쪽 클릭 메뉴

버튼을 오른쪽 클릭하면 자주 사용하는 작업 메뉴가 나타납니다.

- **누르기(Press)** 및 **액션 중단(Abort Actions)** - 버튼을 누르거나 실행 중인 해당 버튼의 액션을 중단합니다.
- **복사(Copy)** 및 **잘라내기(Cut)** - 버튼을 클립보드에 넣습니다.
- **여기에 붙여넣기(Paste here)** 또는 잘라낸 뒤의 **여기로 이동(Move here)** - 클립보드 내용을 현재 버튼 위치에 배치합니다.
- **이 위치와 맞바꾸기(Swap here)** - 현재 버튼과 클립보드의 버튼을 맞바꿉니다.
- **참조로 붙여넣기(Paste as reference)** - 복사한 버튼을 그대로 따르는 버튼 참조를 현재 위치에 만듭니다.
- **비우기(Clear)** - 현재 버튼을 비웁니다.

여러 버튼을 선택한 영역 안에서 오른쪽 클릭하면 **버튼 4개 복사(Copy 4 buttons)** 처럼 선택한 전체 버튼에 메뉴 작업이 적용됩니다.

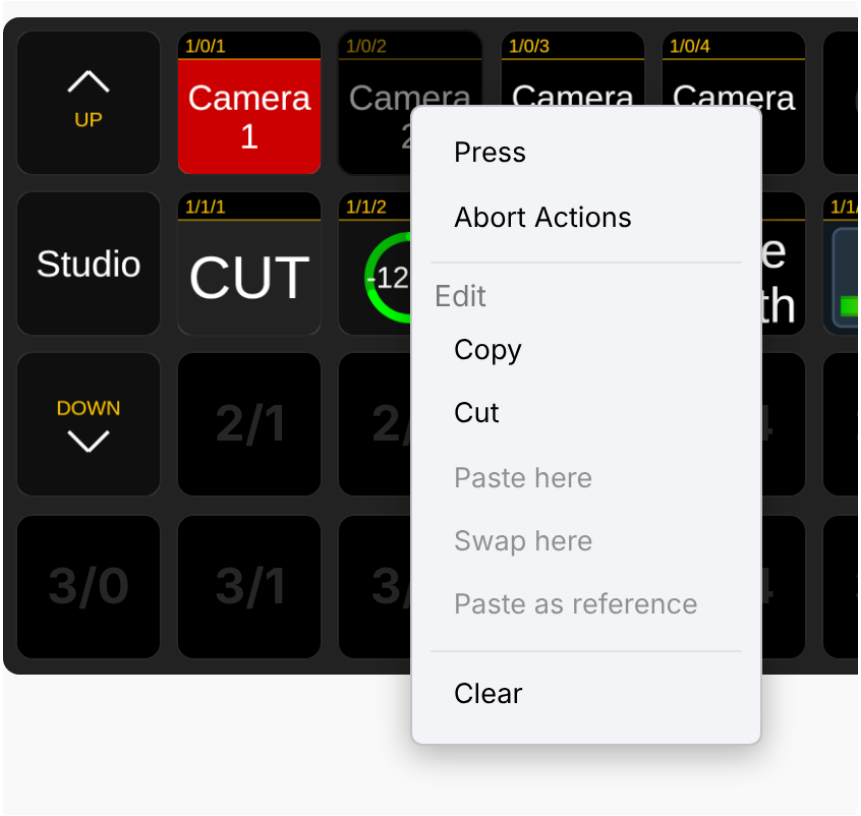


그림 2-12. 그리드 오른쪽 클릭 메뉴

키보드 단축키

그리드에 포커스가 있으면 키보드로도 조작할 수 있습니다.

키	동작
방향키	선택 위치 이동
Shift + 방향키	선택 범위 확장
Ctrl/Cmd + 방향키	선택을 바꾸지 않고 포커스 이동
Space	포커스가 있는 버튼을 선택에 추가하거나 선택에서 제외
Ctrl/Cmd + A	페이지의 모든 버튼 선택
Ctrl/Cmd + C / X / V	선택한 버튼 복사, 잘라내기, 붙여넣기
Delete / Backspace	선택한 버튼 비우기. 실행 전에 확인을 요청합니다.
Page Up / Page Down	다음/이전 페이지로 이동
Ctrl/Cmd + = / - / 0	그리드 확대, 축소, 기본 크기로 복원
Escape	현재 도구, 선택, 클립보드 순서로 취소

여러 버튼을 붙여 넣으면 복사한 선택 영역의 왼쪽 위 버튼을 기준으로 상대적 위치가 유지됩니다.
버튼 중 하나라도 그리드 밖에 놓이게 되면 아무것도 붙여 넣지 않습니다.

2.4 버튼 만들기

버튼을 만드는 방법은 두 가지입니다.

프리셋 사용하기

프리셋을 사용하면 가장 빠르게 버튼을 만들 수 있습니다.

프리셋은 텍스트, 액션, 피드백이 미리 설정된 버튼입니다. 처음부터 모든 항목을 만들 필요 없이 버튼 배치 화면에 끌어 놓으면 됩니다. 모든 모듈이 프리셋을 제공하는 것은 아니며, 액션을 직접 정의하면 더 다양한 동작을 만들 수 있습니다. 그래도 프리셋은 원하는 버튼을 만들기 위한 좋은 출발점이 됩니다.

프리셋은 버튼 그리드 오른쪽의 **프리셋(Presets)** 탭에 있습니다.

사용 중인 모듈이 프리셋을 지원하면 아래처럼 이 탭에 표시되어 선택할 수 있습니다.

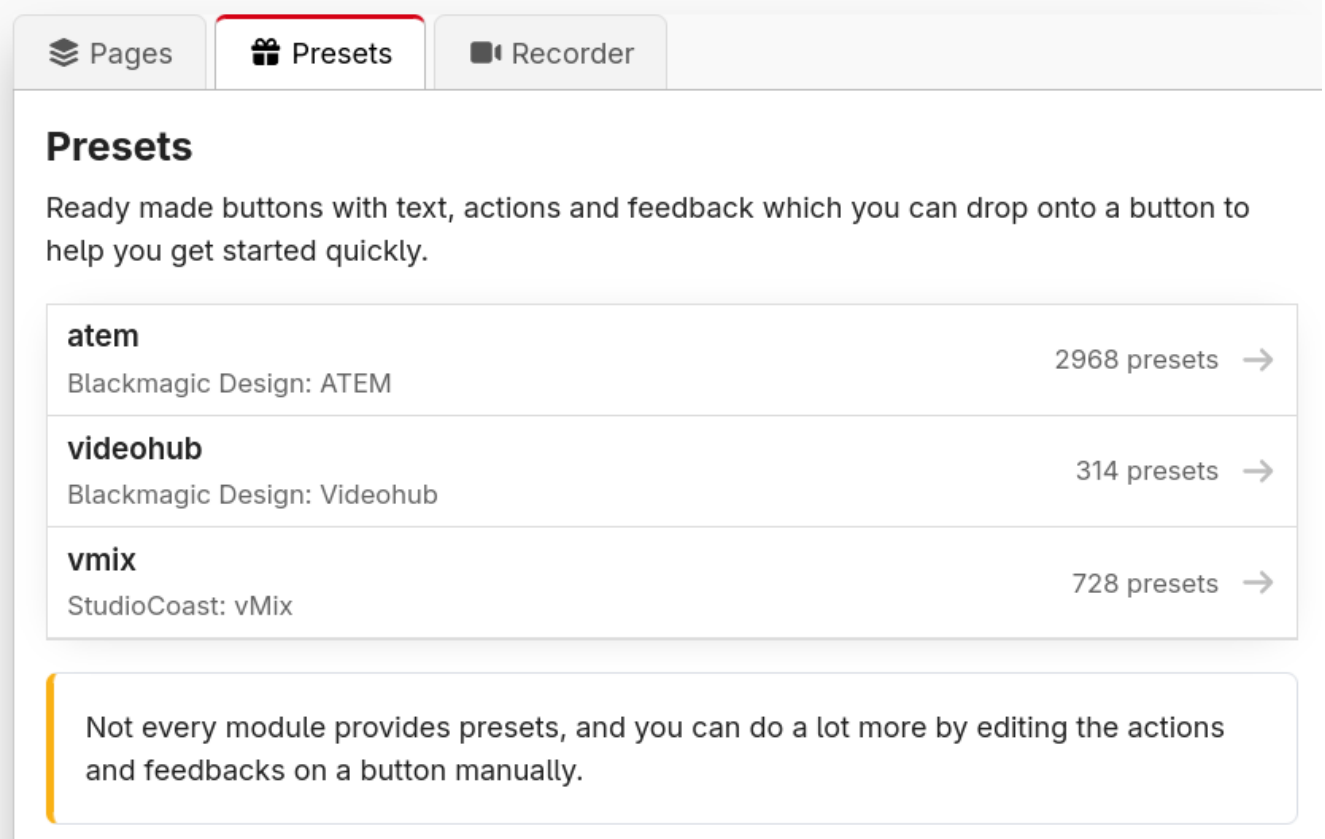


그림 2-13. 프리셋이 준비된 모듈 목록 예

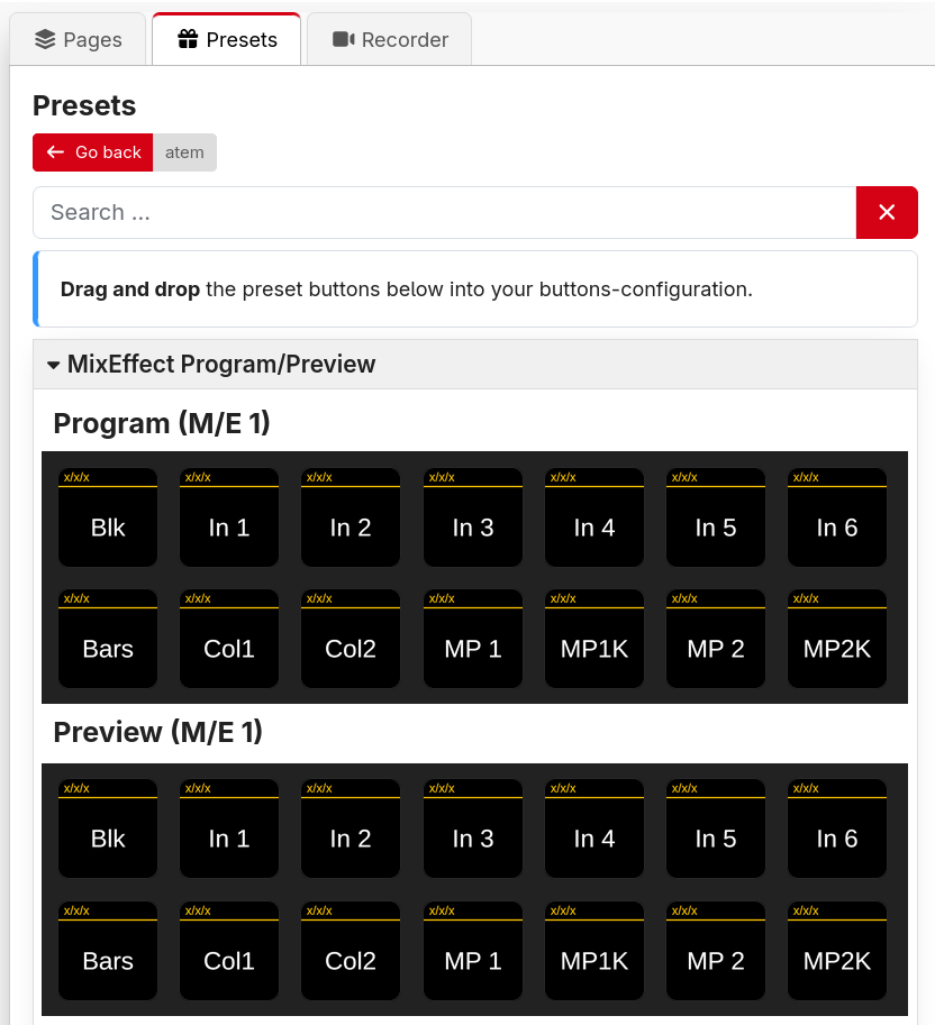


그림 2-14. ATEM ME1용 프리셋 예

버튼 배치(Button Layout) 화면에서 프리셋 버튼을 페이지의 버튼 위치로 끌어 놓습니다.

도움말

그룹 제목을 클릭하면 펼치거나 접을 수 있습니다. **Alt**를 누른 채 클릭하면 같은 단계의 모든 그룹을 한꺼번에 펼치거나 접습니다.

프리셋 연결 또는 복사

프리셋은 두 가지 방식으로 배치할 수 있습니다. Presets 탭 위쪽의 **배치 시(When placed)** 전환 버튼에서 선택합니다.

- **연결(Linked)** - 기본값입니다. 버튼이 프리셋에 연결된 상태를 유지합니다. 모듈을 업데이트하거나 연결 설정을 바꾸는 등의 이유로 모듈이 프리셋을 갱신하면 버튼도 함께 갱신됩니다.
- **복사(Copy)** - 프리셋을 한 번 복사해 버튼을 만듭니다. 직접 만든 버튼처럼 자유롭게 편집할 수 있으며, 이후 프리셋이 바뀌어도 영향을 받지 않습니다.

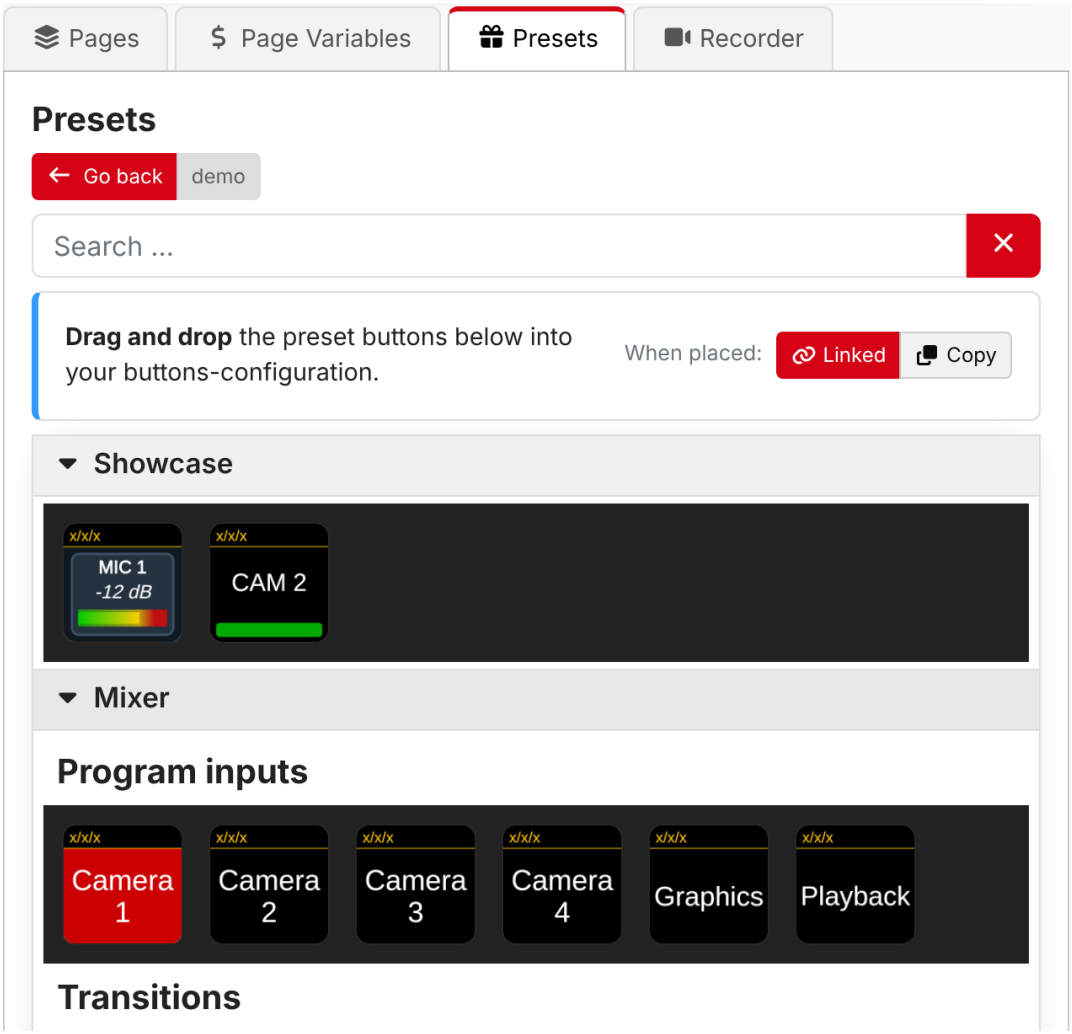


그림 2-15. 프리셋 배치 방식 선택

선택한 방식은 브라우저에 저장되며, 이후에 배치하는 프리셋에만 적용됩니다.

연결된 버튼도 일반 버튼처럼 액션을 실행하고 피드백을 표시하며 여러 단계를 사용할 수 있습니다.
다만 구성은 프리셋이 관리하므로 직접 바꿀 수 없습니다. 연결된 버튼을 선택하면 편집기에는 다음 항목이 표시됩니다.

- **연결(Connection)** - 버튼이 사용할 연결입니다. 같은 모듈을 사용하는 다른 연결로 바꿀 수 있습니다.
  예를 들어 여러 버튼이 두 번째 영상 스위처를 제어하도록 바꿀 수 있습니다.
- **프리셋 변수(Preset variables)** - 스위처 입력마다 버튼 하나씩을 만드는 것처럼, 일부 프리셋은 비슷한 버튼 그룹의 템플릿 역할을 합니다. 버튼마다 다른 부분을 결정하는 값이 여기에 표시되며 수정할 수 있습니다.

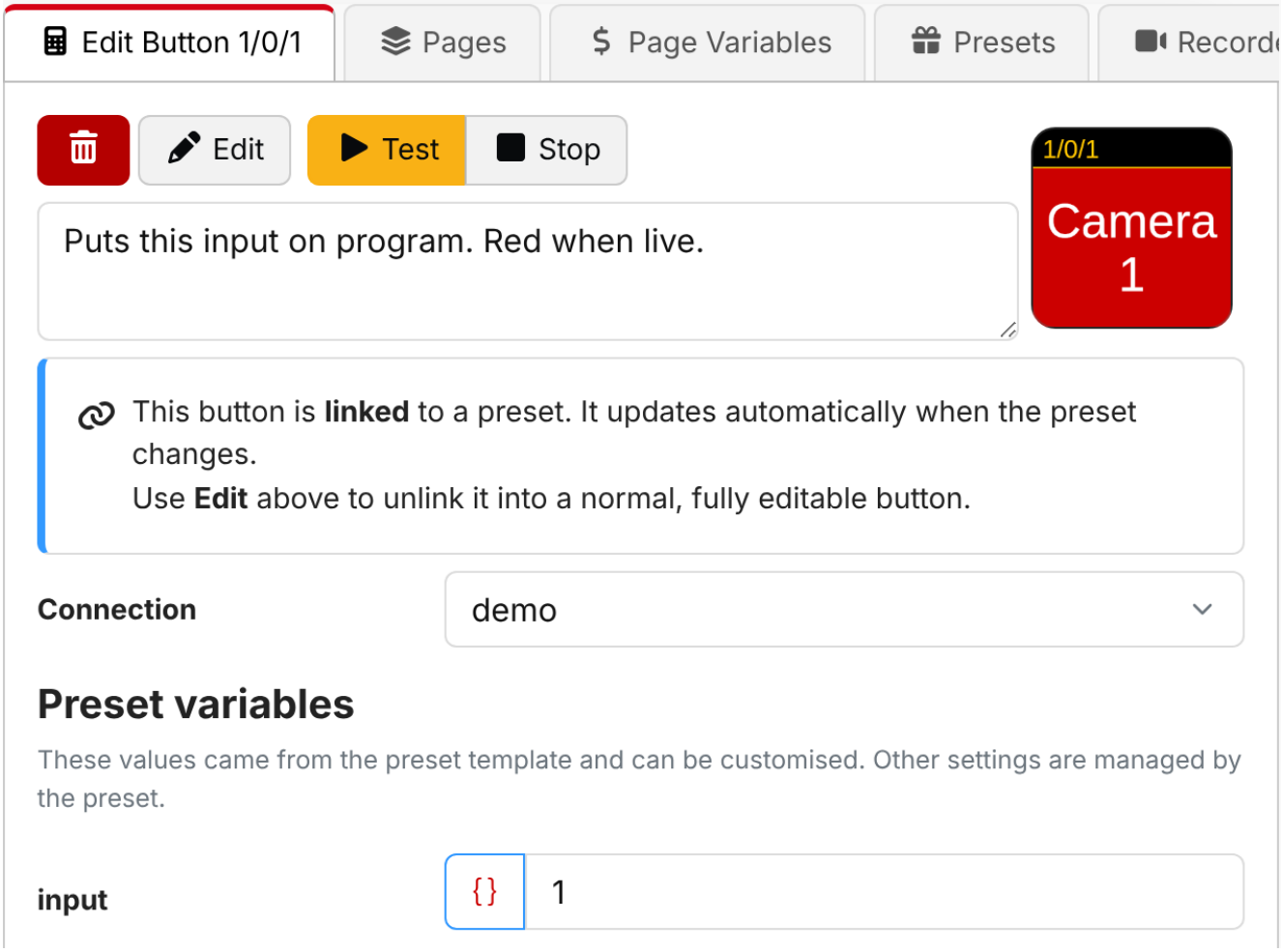


그림 2-16. 연결된 프리셋 버튼

연결된 버튼을 더 자세히 수정하려면 편집기 위쪽의 **편집(Edit)** 을 클릭합니다. 프리셋과의 연결이 해제되고 같은 모양·액션·피드백을 가진 일반 버튼으로 바뀌므로 자유롭게 수정할 수 있습니다. 이 작업은 되돌릴 수 없으며, 이후 프리셋 업데이트도 더 이상 적용되지 않습니다.

연결된 버튼이 사용하는 연결을 비활성화하거나 삭제하면 버튼은 마지막으로 표시했던 모습을 유지합니다.

참고

연결된 프리셋을 사용하려면 모듈 API 2.0 이상으로 작성된 모듈이 필요합니다. 이전 모듈에서는 전환 버튼이 비활성화되며 프리셋이 항상 복사 방식으로 배치됩니다.

참고

프리셋은 모듈 제작자가 미리 만드는 것이므로 사용자가 직접 만들 수는 없습니다. 대신 다른 페이지에 자신만의 프리셋용 버튼 모음을 구성하고 내보내거나 다시 가져올 수 있습니다.

추가 프리셋이 필요하다면 해당 모듈의 GitHub Issues 페이지에 요청할 수 있습니다. 링크는 연결 페이지에 있습니다.

직접 만들기

1. 버튼을 만들 위치의 빈 버튼을 클릭합니다.
2. 만들 버튼의 유형을 선택합니다.
   - **일반 버튼(Regular button)**: 하나 이상의 액션을 실행합니다. 가장 자주 사용하는 유형입니다.
   - **버튼 참조(Button reference)**: 다른 버튼을 그대로 따릅니다. 아래의 버튼 참조를 참고하십시오.
   - **페이지 위로(Page up)**: 제어 장치를 다음 페이지로 이동합니다.
   - **페이지 번호(Page number)**: 현재 페이지의 번호와 이름을 표시합니다. 누르면 제어 장치가 **시작 페이지**로 돌아갑니다.
   - **페이지 아래로(Page down)**: 제어 장치를 이전 페이지로 이동합니다.

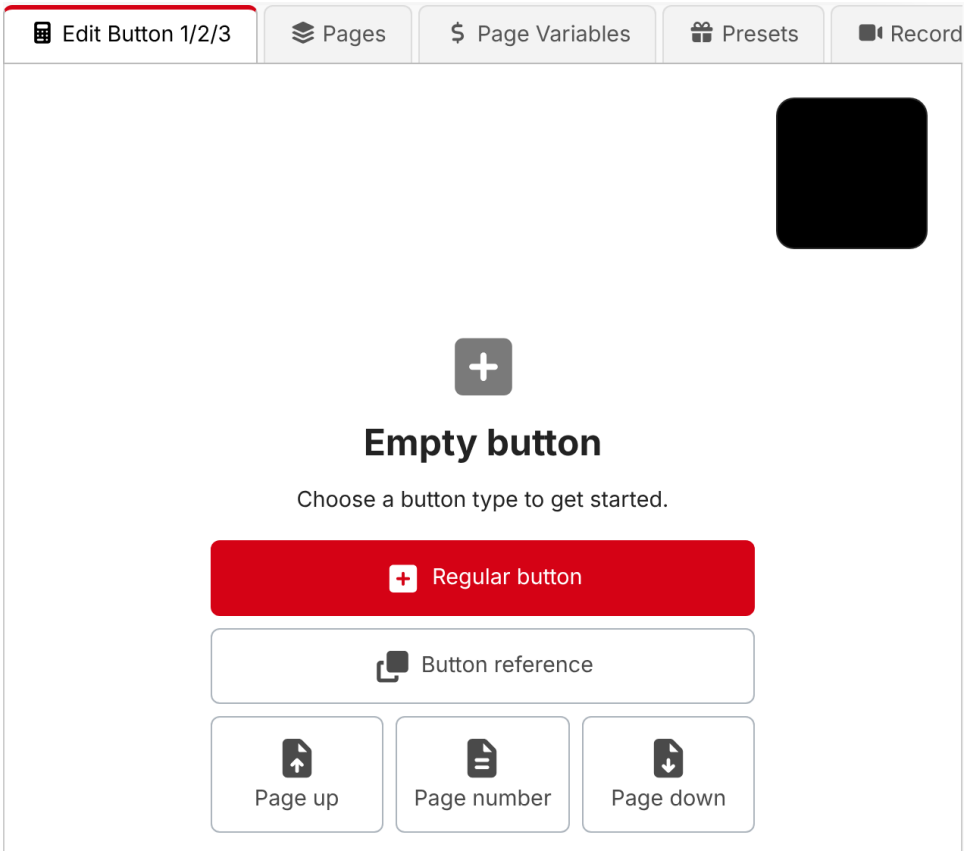


그림 2-17. 버튼 유형 선택

페이지 이동 버튼 수정하기

Page up, **Page number**, **Page down**은 정해진 동작이 내장된 특수 버튼입니다. 모양이나 동작을 바꾸려면 **편집(Edit)**, 즉 연필 아이콘을 클릭해 **일반 버튼으로 변환(Convert to Normal Button)** 합니다. 같은 동작의 액션이 미리 채워진 일반 버튼으로 바뀌므로 모양과 동작을 자유롭게 수정할 수 있습니다.

일반 버튼으로 변환한 뒤에는 되돌릴 수 없습니다.

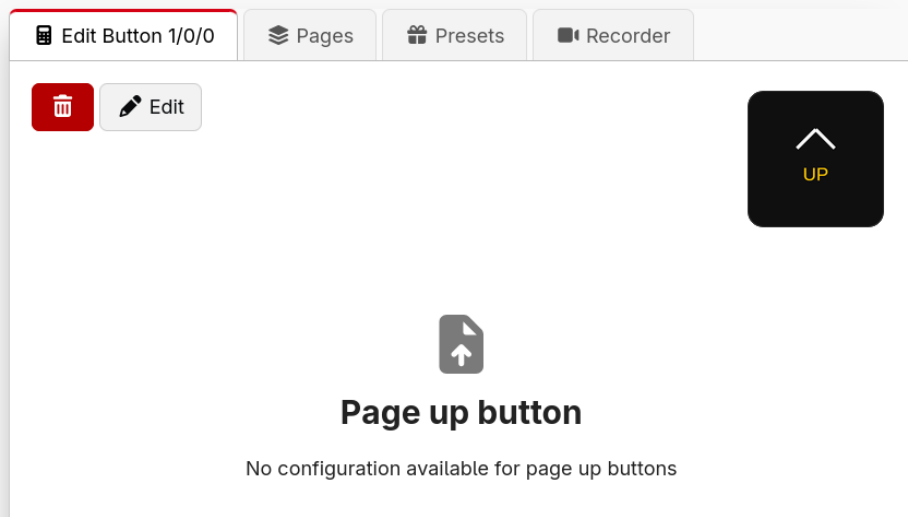


그림 2-18. 일반 버튼으로 변환

버튼 모양 설정하기

버튼의 모양은 텍스트, 이미지, 도형 등의 그래픽 요소를 여러 겹 쌓아 만듭니다. 자세한 내용은 버튼 스타일에서 설명합니다.

편집기 위쪽 모서리에 버튼의 실시간 미리보기가 표시됩니다. 변경 내용은 에뮬레이터와 연결된 제어 장치에도 즉시 반영되어 확인할 수 있습니다.

동작 옵션

버튼에는 모양 외에도 다음 동작 옵션이 있습니다.

- **단계 진행(Step progression)** - 단계가 여러 개일 때 다음 단계로 넘어가는 방식입니다. 단계를 참고하십시오.
- **회전 액션(Rotary actions)** - 켜면 Stream Deck +나 회전 기능이 있는 다른 제어 장치의 다이얼·인코더를 사용할 수 있습니다. 아래의 회전 액션을 참고하십시오.

액션 추가하기

누름/켜짐 액션 추가(Add Press/on action) 드롭다운 메뉴에서 버튼에 액션을 추가합니다. 여러 액션을 넣고 **internal: Wait** 액션으로 실행 시점을 지연시킬 수 있습니다. 지연 시간의 단위는 밀리초이며, 1000ms = 1초입니다.
자세한 내용은 액션을 참고하십시오.

회전 액션

Rotary actions를 켜면 버튼에 **왼쪽 회전 액션(Rotate left actions)** 과 **오른쪽 회전 액션(Rotate right actions)** 그룹이 추가됩니다. 이 버튼에 해당하는 다이얼이나 인코더를 돌리면 실행됩니다.

많은 제어 장치는 다이얼을 한 번 움직일 때 얼마나 돌렸는지도 보고합니다. 특히 빠르게 돌릴 때 유용합니다. 회전 액션은 다이얼을 움직일 때마다 한 번 실행되며, $(this:delta) 변수가 회전량을 알려 줍니다. 오른쪽, 즉 시계 방향은 양수이고 왼쪽은 음수입니다. 예를 들어 값을 $(this:delta)만큼 변경하는 액션을 사용하면 다이얼을 빠르게 돌릴수록 값도 빠르게 바뀝니다.

$(this:delta)는 회전 액션 안에서만 사용할 수 있습니다.

Button: Trigger rotate 액션은 다른 버튼의 다이얼을 돌린 것처럼 회전 동작을 실행합니다. **회전량(Amount)** 은 회전 단계 수이며, 양수는 오른쪽, 음수는 왼쪽입니다. 여기에 $(this:delta) 표현식을 지정하면 실제 장치의 회전 동작을 다른 버튼으로 전달할 수 있습니다.

버튼 참조

버튼 참조(Button reference) 는 다른 버튼을 그대로 따릅니다. 원본과 똑같이 표시되며, 누르거나 돌리면 원본 버튼을 누르거나 돌린 것처럼 작동합니다. 예를 들어 모든 페이지에 같은 "컷" 버튼을 두고 싶을 때 유용하며, 복사본을 각각 수정해 맞출 필요가 없습니다.

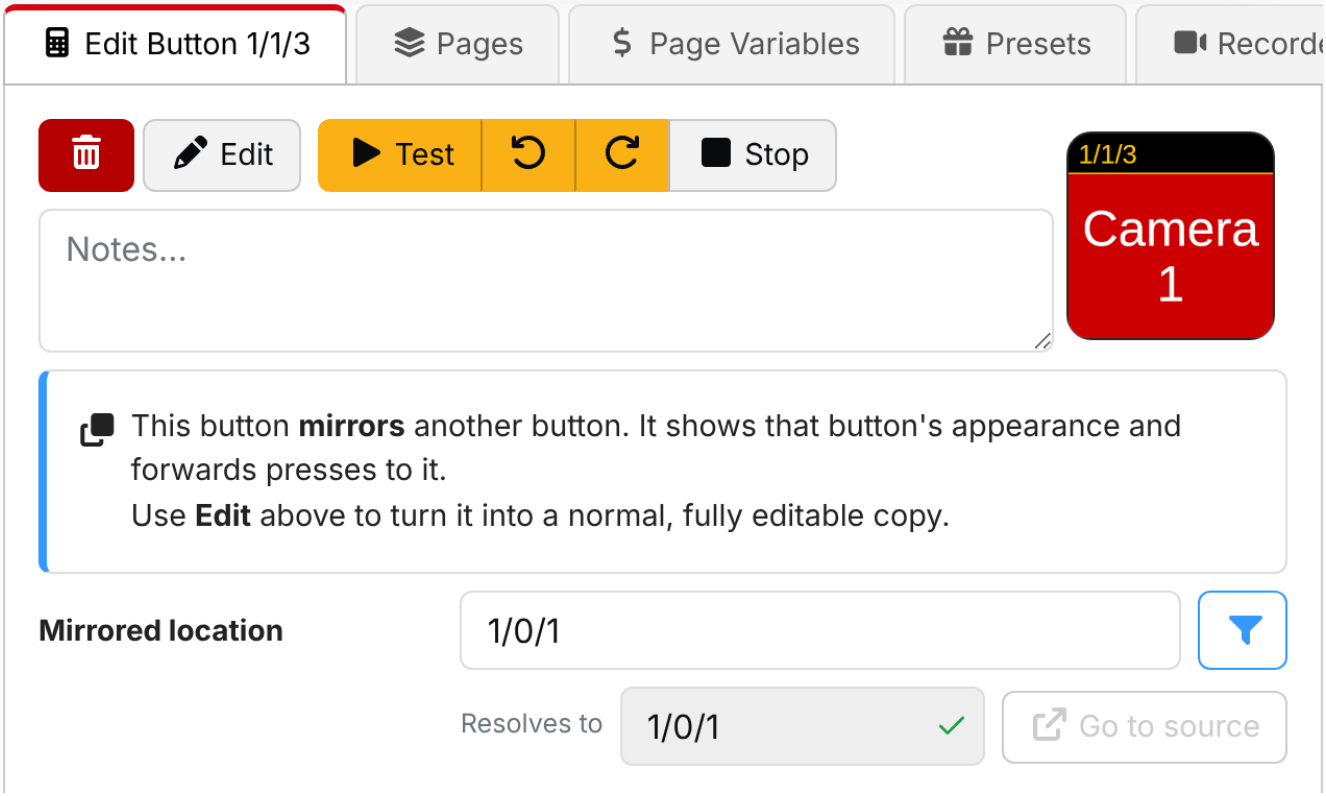


그림 2-19. 버튼 참조

빈 버튼을 만들 때 **Button reference**를 선택하거나, 버튼을 복사한 뒤 참조를 놓을 위치에서 오른쪽 클릭하고 **참조로 붙여넣기(Paste as reference)** 를 선택합니다.

참조의 설정 항목은 **참조 위치(Mirrored location)** 하나뿐입니다. 1/0/1처럼 원본 버튼의 위치를 page/row/column 형식으로 지정합니다. 변수나 표현식을 사용할 수 있으므로 다른 조건에 따라 참조할 버튼을 바꿀 수도 있습니다. 예를 들어 페이지 번호를 저장하는 camera 페이지 변수가 있다면 $(page:camera)/0/1은 해당 페이지의 행 0, 열 1 버튼을 참조합니다. **해석된 위치(Resolves to)** 에는 현재 참조 중인 위치가 표시되며, **원본으로 이동(Go to source)** 을 클릭하면 그 버튼이 편집기에 열립니다.

- 유효하지 않은 버튼 위치이거나 해당 버튼이 비어 있으면 "Unresolved Reference"가 표시됩니다.

- 직접 또는 다른 참조를 거쳐 자기 자신을 가리키게 되면 "∞"가 표시됩니다.

편집기 위쪽의 **Edit**을 클릭하면 참조 중인 버튼의 편집 가능한 일반 복사본으로 바뀝니다. 이 작업은 되돌릴 수 없습니다.

참고

페이지를 가져올 때 버튼 참조의 위치는 자동으로 조정되지 않습니다. 가져온 페이지의 번호가 달라지면 참조를 수정해야 할 수 있습니다.

메모

모든 버튼에는 **메모(Notes)** 필드가 있습니다. 버튼 편집기 위쪽의 Notes... 상자를 찾으십시오. 여기에 입력한 내용은 사용자를 위한 내부 메모이며 버튼 자체에는 표시되지 않습니다. 버튼을 그렇게 설정한 이유를 적어 두면 몇 달 뒤 복잡한 공연 설정을 다시 살펴볼 때 도움이 됩니다. 트리거와 표현식 변수에도 같은 메모 필드가 있습니다.

일부 프리셋은 배치될 때 버튼의 기능을 설명하는 메모를 자동으로 채웁니다. 이 메모는 자유롭게 수정할 수 있습니다.

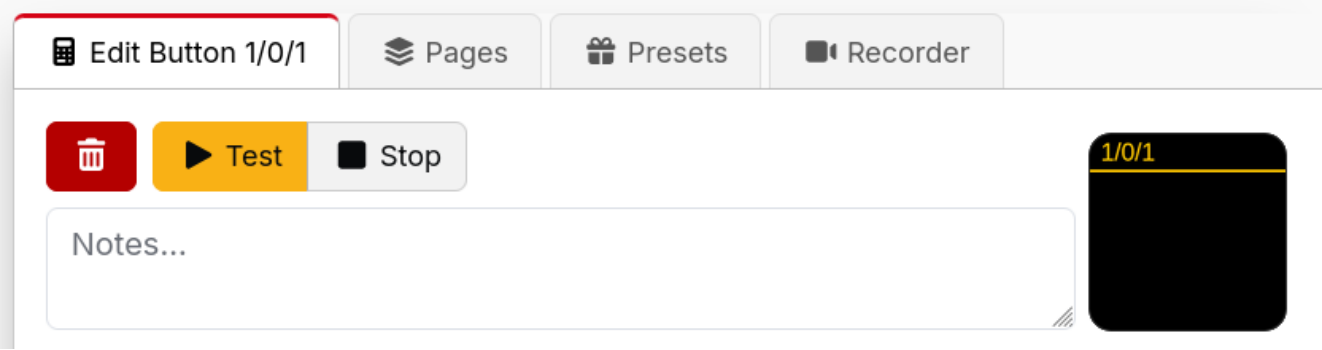


그림 2-20. 버튼 메모

2.5 버튼 스타일

Companion 5.0에서는 버튼을 그리는 방식이 완전히 바뀌었습니다. 하나의 고정된 스타일, 즉 텍스트 한 묶음·배경 이미지 하나·색상 한 세트를 설정하는 대신 **그래픽 요소**를 여러 겹 쌓아 버튼을 만듭니다. 텍스트, 이미지, 도형 같은 작은 구성 요소를 겹쳐 원하는 모양을 만들 수 있습니다.

도움말 이전 Companion을 사용했다면

기존 버튼 스타일에서 설정하던 텍스트, 글자 크기, 배경 이미지, 색상은 모두 그대로 사용할 수 있습니다.
다만 각각 추가·삭제·이동하거나 모양을 바꿀 수 있는 별도의 **요소**로 나뉘었습니다. 예를 들어 텍스트와 배경 이미지가 있는 단순한 버튼은 이제 이미지 요소와 텍스트 요소, 두 개로 구성됩니다. 기존 기능은 그대로이며 더 다양한 표현이 가능해졌습니다.

이전의 **배경색**은 이제 요소 목록 아래쪽의 **배경(Background)** 요소가 담당합니다. 별도의 "배경" 유형이 있는 것은 아닙니다.
테두리 없이 버튼 전체를 채우도록 설정된 **상자(Box)** 요소입니다(X 0, Y 0, Width 100, Height 100).
다른 요소처럼 모양이나 이름을 바꾸고 삭제할 수 있습니다. 삭제한 뒤 다시 필요해지면 **Box** 요소를 추가하십시오.

레이어 편집기의 구성

버튼의 **스타일(Style)** 탭을 열면 모양을 편집할 수 있습니다. 위쪽에는 다음 두 영역이 있습니다.

- 실시간 **미리보기** - 수정 내용이 즉시 표시되며 실제 제어 장치와 에뮬레이터에도 실시간으로 반영됩니다.
  미리보기에서 요소를 직접 선택·이동하거나 크기를 바꿀 수 있습니다. 미리보기에서 편집하기를 참고하십시오.
- **요소 목록** - 버튼을 구성하는 요소가 쌓인 순서입니다. 이미지 편집기의 레이어처럼 목록 위쪽의 요소가
  아래쪽 요소 위에 그려집니다.

그 아래에는 선택한 항목의 속성이 표시됩니다. 두 영역 사이의 막대를 드래그해 어느 쪽에 더 많은 공간을
줄지 조절할 수 있습니다.

목록 아래에는 레이어가 아닌 다음 두 항목의 버튼이 있습니다.

- **고정 속성(Pinned)** - 버튼의 여러 요소에서 자주 편집하는 속성을 모아 놓습니다. 아래의 고정 속성을 참고하십시오.
- **캔버스(Canvas)** - 항상 존재하며 삭제할 수 없습니다. 상단 표시줄, 상태 아이콘, 빈 버튼의 모양을 제어합니다.
  캔버스와 버튼 표시 정보를 참고하십시오.

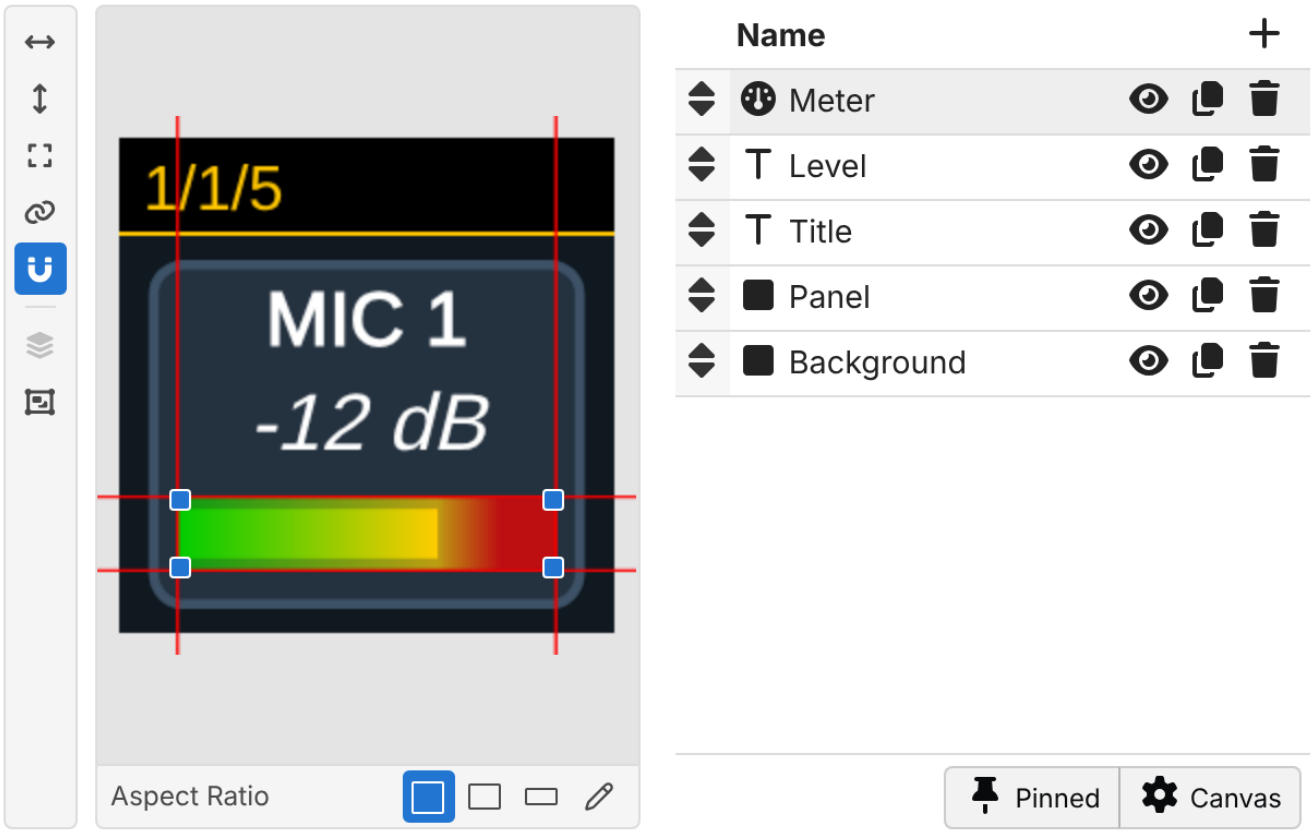


그림 2-21. 레이어 방식 버튼 편집기

요소 추가하기

요소 목록 위쪽의 **+** (Add element, 요소 추가)를 클릭하고 유형을 선택합니다. 새 요소는 선택한 요소 바로 위에 추가됩니다. 기본으로 제공되는 유형은 다음과 같습니다.

요소	용도
그룹(Group)	다른 요소를 담아 함께 이동하거나 재사용하는 컨테이너
텍스트(Text)	이름, 변수, 표현식 결과 등의 텍스트 표시
이미지(Image)	컴퓨터에서 올리거나 이미지 라이브러리에서 선택한 이미지 표시
상자(Box)	채워진 사각형. 둥근 모서리와 테두리를 선택적으로 설정 가능
선(Line)	두 점을 잇는 직선
원(Circle)	원 또는 호. 테두리를 선택적으로 설정 가능
게이지(Gauge)	레벨 미터나 진행 표시처럼 값을 나타내는 막대 또는 링
참조(Reference)	다른 버튼의 그래픽을 그대로 표시. 디자인 재사용에 유용

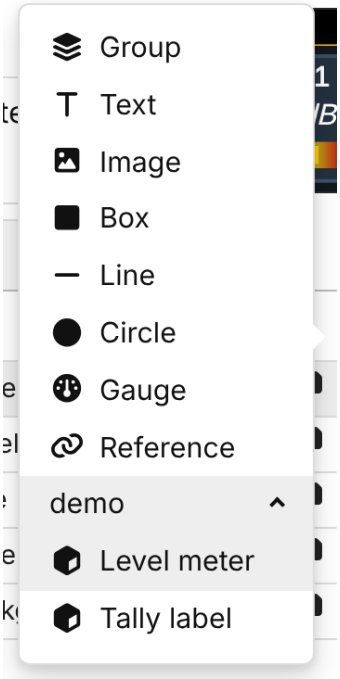


그림 2-22. 요소 추가 메뉴

일부 모듈은 장치 전용 미터나 상태 표시 같은 미리 만들어진 **복합 요소(composite element)** 도 제공합니다.
연결이 복합 요소를 제공하면 **Add element** 메뉴 아래쪽에 해당 연결 이름으로 묶여 표시됩니다.
이름을 클릭해 목록을 펼치고 요소 위에 마우스를 올리면 설명을 볼 수 있습니다.

복합 요소는 하나의 단위로 그려집니다. 이동, 크기 조정, 회전과 **속성(Properties)** 영역에서 모듈이 제공하는 옵션 변경은 가능하지만, 내부의 개별 도형은 수정할 수 없습니다. 복합 요소를 제공하는 연결이 비활성화되거나 삭제되면 연결이 복구될 때까지 "Missing Element"로 표시됩니다.

참고 앞으로 추가될 요소

현재의 기본 요소는 시작 단계입니다. 공통으로 필요한 기능을 파악하면서 구성 요소를 추가할 예정이므로 이후 버전에서는 목록이 늘어날 수 있습니다. 유용할 만한 요소 유형이 있다면 알려 주십시오.

순서 변경·복제·삭제

요소 목록의 각 행에는 다음 조작 기능이 있습니다.

- **드래그 핸들(Drag handle)** - 요소를 위아래로 드래그해 쌓이는 순서를 바꿉니다. 그룹 안이나 밖으로 옮길 수도 있습니다.
- **표시 여부(Visibility)** - 미리보기에서 요소를 표시하거나 숨깁니다.
- **복제(Duplicate)** - 모든 속성이 같은 요소를 복사합니다. 비슷한 텍스트 두 개나 일정한 간격의 상자 여러 개를 만들 때 유용합니다.
- **삭제(Remove)** - 요소를 삭제합니다.

미리보기에서 편집하기

그림 프로그램처럼 미리보기에서 버튼의 배치를 직접 편집할 수 있습니다.

- 요소를 **클릭**해 선택합니다. 마우스 포인터 아래에서 가장 위에 있는 요소가 선택됩니다. 미리보기의 빈 곳을 클릭하면 선택이 해제됩니다.
- 선택한 요소를 **드래그**해 이동합니다.
- **모서리 핸들을 드래그**해 크기를 바꿉니다.
- **선(Line)** 은 모서리 대신 양 끝에 핸들이 있으며, 드래그하면 해당 끝점을 이동합니다.

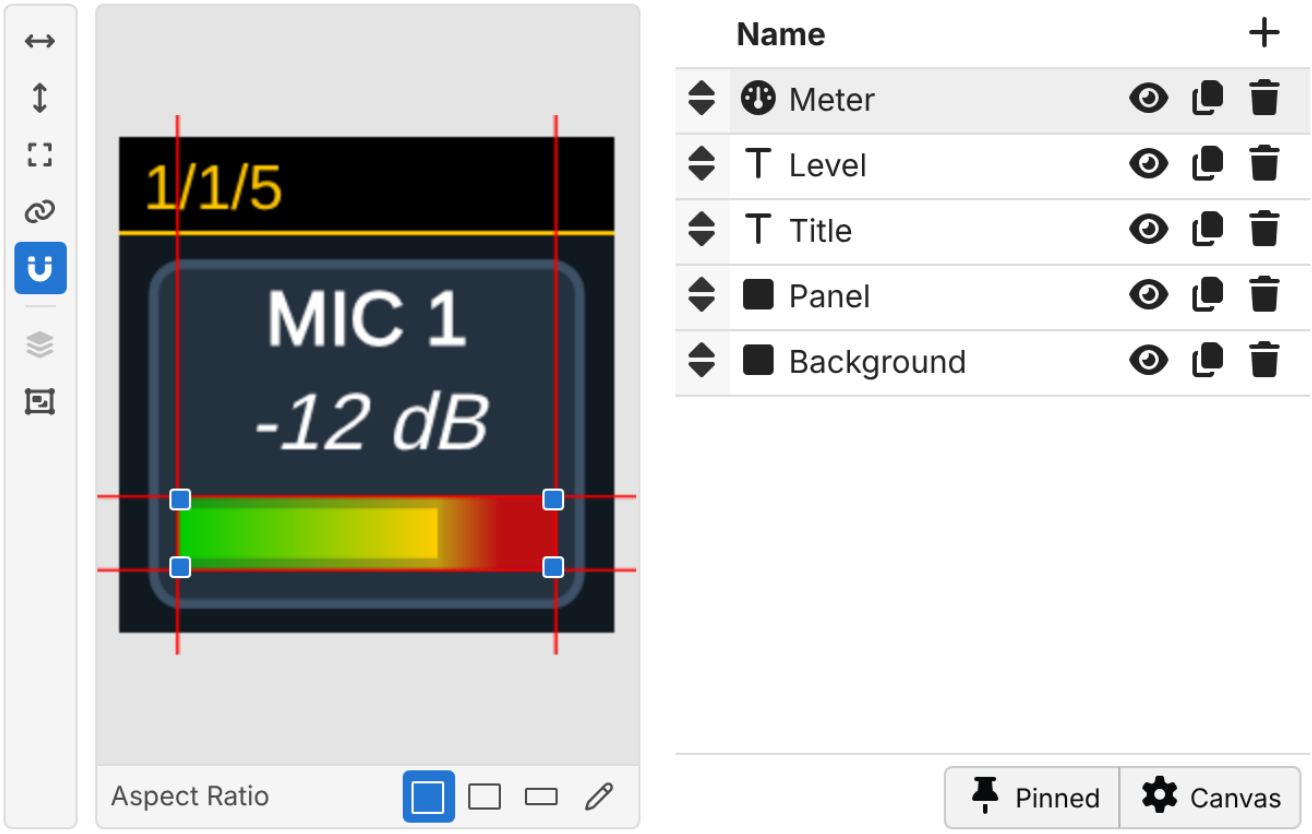


그림 2-23. 미리보기에서 편집하기

드래그할 때 다음 키를 함께 사용할 수 있습니다.

- **Shift** - 가로 또는 세로로만 움직입니다. 선은 정확히 수평이나 수직으로 맞춰집니다.
- **Alt** - 요소를 복제한 뒤 복사본을 드래그합니다.
- **Ctrl**(macOS에서는 **Cmd**) - 스냅을 일시적으로 켜거나 끕니다.

미리보기 옆의 도구 모음에는 선택한 요소에 빠르게 적용할 수 있는 기능이 있습니다.

- **가로 가운데 정렬(Center horizontally)** 및 **세로 가운데 정렬(Center vertically)** - 버튼의 가운데로 정렬합니다.
- **채우기(Fill)** - 요소가 버튼 전체를 채우도록 합니다.
- **비율 유지(Link)** - 크기를 바꿀 때 너비와 높이의 비율을 유지합니다.
- **스냅(Snapping)** - 드래그하는 동안 버튼과 다른 요소의 가장자리·중앙에 맞춰 붙입니다. 안내선이 맞춰진 위치를 표시합니다. 기본적으로 켜져 있습니다.
- **맨 앞으로(Bring to front)** 및 **맨 뒤로(Send to back)** - 요소를 목록의 맨 위 또는 맨 아래로 옮깁니다.

일부 요소는 미리보기에서 편집할 수 없습니다. 비활성화된 도구 버튼에 마우스를 올리면 이유를 확인할 수 있습니다.
그룹 안의 요소나 위치·크기가 표현식으로 설정된 요소가 이에 해당합니다. 이런 요소는 아래쪽 속성에서 편집하십시오.

다른 가로세로 비율 미리보기

모든 제어 장치의 버튼이 정사각형인 것은 아닙니다. 미리보기 아래의 **가로세로 비율(Aspect Ratio)** 버튼을 사용하면 장치를 연결하지 않고도 다른 모양의 버튼에서 디자인이 어떻게 보이는지 확인할 수 있습니다.

- **1:1** - 대부분의 Stream Deck처럼 정사각형입니다.
- **9:7** - Stream Deck Studio의 버튼 비율입니다.
- **2:1** - Stream Deck +에서 각 다이얼 위에 있는 화면의 비율입니다.
- 연필 아이콘으로 다른 **W**(너비)와 **H**(높이)를 입력할 수 있습니다. Companion이 인식했던 제어 장치의 버튼 모양도 목록에 나타나므로 쉽게 선택할 수 있습니다.

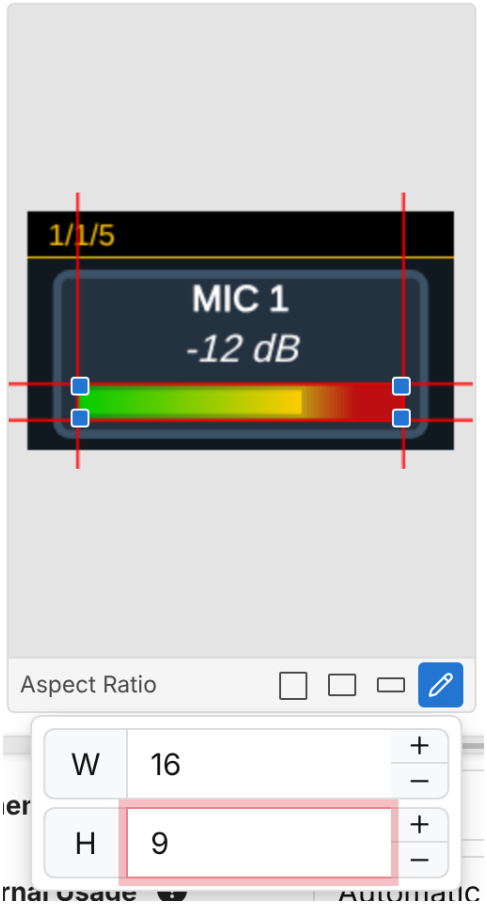


그림 2-24. 사용자 지정 가로세로 비율

이 설정은 편집기의 미리보기에만 영향을 주며 브라우저에 저장됩니다. 위치와 크기는 버튼에 대한 백분율로 지정하므로 같은 디자인을 여러 모양에 사용할 수 있습니다. 아래의 위치와 크기를 참고하십시오.

도움말

위치와 크기가 백분율이므로 도형은 버튼 크기에 맞춰 늘어납니다. 화살표나 아이콘처럼 버튼 모양이 달라도 형태를 유지해야 하는 부분은 정사각형 비율을 켠 그룹에 넣으십시오. 그러면 버튼 모양과 관계없이 그룹 중앙의 정사각형 안에 그려집니다.

요소 속성

요소를 선택해 속성을 편집합니다. 자주 바꾸는 몇 가지 속성만 필요하다면 요소를 선택하는 대신 아래의 고정 속성을 사용하십시오.

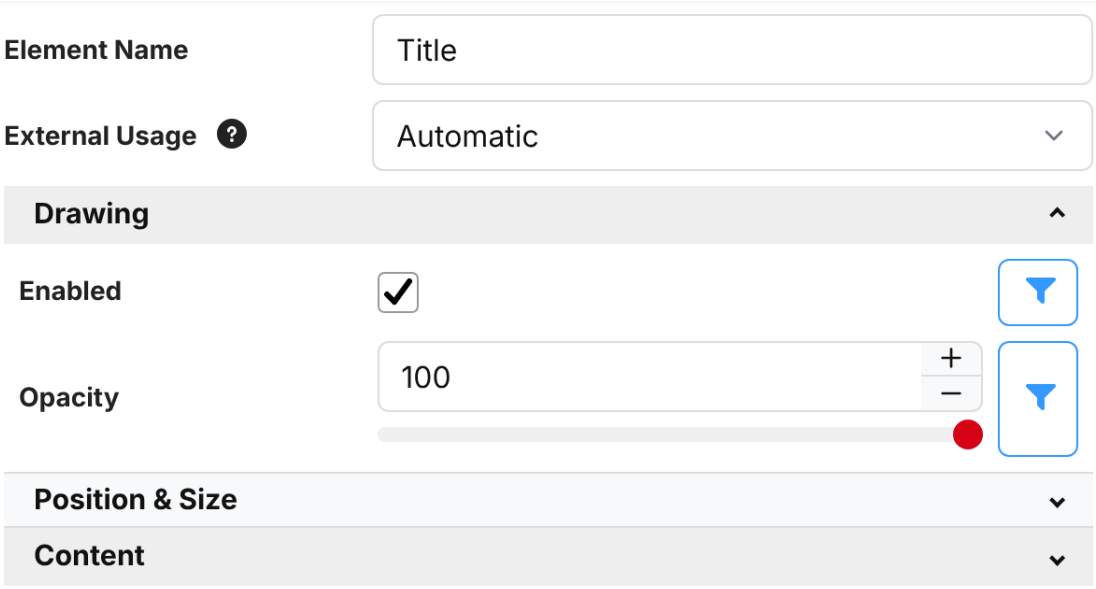


그림 2-25. 요소 속성 영역

위쪽에는 모든 요소에 공통으로 있는 두 필드가 있습니다.

- **요소 이름(Element Name)** - 요소 목록에 표시할 이름입니다. 피드백의 스타일 덮어쓰기에서도 이 이름으로 요소를 고르므로 알아보기 쉽게 지정하십시오.
- **외부 사용(External Usage)** - 일부 제어 장치는 컬러 화면 대신 색상 LED나 링 모양 표시등으로 버튼 상태를 간단하게 표시합니다. Companion은 보통 여기에 사용할 요소를 자동으로 선택합니다. 잘못된 요소가 선택되면 이 항목을 **텍스트(Text)**, **색상(Color)**, **이미지(Image)** 또는 **게이지(Gauge)** 로 설정해 해당 용도로 사용할 요소를 지정하십시오. 제어 장치의 LED에 게이지 표시하기를 참고하십시오.

나머지 속성은 패널을 보기 쉽게 접을 수 있는 **영역**으로 나뉩니다. 캔버스를 제외한 거의 모든 요소가 다음 두 영역을 공유합니다.

그리기

- **사용(Enabled)** - 요소를 그릴지 정합니다. 표현식을 사용할 수 있으므로 조건에 따라 표시하거나 숨길 수 있습니다. 예를 들어 blink(1000)으로 설정하면 1초마다 깜박이며, $(custom:armed)로 설정하면 사용자 정의 변수 armed가 참인 동안에만 표시됩니다.
- **불투명도(Opacity)** - 요소의 불투명한 정도로, 0은 보이지 않음, 100은 완전 불투명입니다. 값을 낮추면 아래의 레이어가 비쳐 보입니다.

위치와 크기

요소의 위치는 고정 픽셀이 아닌 버튼에 대한 **백분율**로 정합니다. 따라서 작은 Stream Deck 키와 큰 고해상도 제어 장치에서도 같은 형태로 표시됩니다.

- **X % / Y %** - 요소의 왼쪽 위 위치로, 버튼의 가로·세로에 대한 백분율입니다.
- **Width % / Height %** - 버튼에 대한 요소의 너비·높이 백분율입니다.

- **회전(Rotation (degrees))** - 요소의 중심을 기준으로 회전합니다. 원은 회전해도 차이가 없으므로 이 항목이 없습니다.

선(Line) 에는 대신 **위치(Position)** 영역이 있으며, 선의 **시작점(From)** 과 **끝점(To)** 을 설정합니다.

예를 들어 X 0, Y 0, Width 100, Height 100은 버튼 전체를 채웁니다. X 50, Y 0, Width 50, Height 100은 버튼의 오른쪽 절반을 채웁니다.

속성에서 표현식 사용하기

각 속성 행의 끝에 있는 버튼으로 표현식 입력을 선택할 수 있습니다. 표현식을 사용하면 Companion이 속성 이름 아래에 현재 계산 결과를 표시하므로 의도대로 작동하는지 쉽게 확인할 수 있습니다. 표현식이 올바르지 않거나 실행에 실패하면 경고가 나타납니다.



그림 2-26. 표현식 값 미리보기

버튼의 피드백이 덮어쓰는 속성에는 이름 옆에 작은 레이어 아이콘이 표시됩니다. 여기에 지정한 값이 항상 그대로 표시되는 것은 아니라는 뜻입니다.

고정 속성

여러 버튼의 문구를 바꿀 때마다 요소를 선택하고 원하는 속성까지 스크롤하는 것은 번거롭습니다.
요소 목록에서 **고정 속성(Pinned)** 을 선택하면 버튼 전체에서 고정해 둔 속성이 요소별로 모여 표시됩니다.
요소를 먼저 선택하지 않고도 텍스트, 글자 크기와 색상, 배경색 등을 한 곳에서 편집할 수 있습니다.

고정 기능을 지원하는 요소 유형에는 다음 속성이 기본으로 고정됩니다.

요소	기본 고정 속성
텍스트(Text)	텍스트, Text Size, Shrink to fit, Color, 정렬
이미지(Image)	이미지와 정렬
상자(Box)	색상(Color)
원(Circle)	색상(Color)
선(Line)	선 색상(Line Color)과 선 두께(Line Width)
게이지(Gauge)	값(Value)
참조(Reference)	위치(Location)

그룹과 복합 요소에는 기본 고정 속성이 없습니다. 고정된 속성이 없는 요소는 이 보기에 나타나지 않습니다.

각 그룹의 제목에는 해당 요소가 표시되며, 속성 패널의 영역처럼 접을 수 있습니다. 제목 끝의 화살표인 **모든 속성 편집(Edit all properties)** 을 클릭하면 요소의 전체 속성 패널이 열리므로 고정하지 않은 속성도 한 번의 클릭으로

볼 수 있습니다. 위쪽 요소 목록에서 해당 요소를 선택해도 됩니다. 제목에는 요소가 비활성화되었거나 숨겨진 상태인지도 표시되므로, 표시되지 않는 요소를 불필요하게 편집하는 일을 줄일 수 있습니다.

속성 고정과 해제

요소의 속성 패널에서 항목에 마우스를 올리면 이름 옆에 **핀** 아이콘이 나타납니다. 클릭하면 그 속성을 현재 버튼에 고정합니다. 고정 속성 보기에서 항목에 마우스를 올리면 같은 아이콘으로 고정을 해제할 수 있습니다. 고정 상태는 버튼의 각 요소별로 저장되므로, 한 버튼의 글자 크기를 고정해도 다른 버튼에는 영향을 주지 않습니다.

Companion은 요소 목록에서 마지막으로 선택한 항목을 기억하고 다음 버튼을 열 때도 같은 항목을 선택합니다.
모든 버튼에 **Pinned**가 있으므로 이를 선택해 두면 이후 클릭하는 버튼에서도 고정 속성이 바로 열립니다.

기본 도형 요소

상자(Box), **선(Line)**, **원(Circle)** 은 간단한 도형으로, 배경·구분선·테두리·상태 표시점 등에 유용합니다.

- **Box** - 채워진 사각형입니다. **채우기(Fill)** 와 **테두리(Border)** 영역이 있으며, 테두리에서는 두께·색상과 가장자리의 안쪽/중앙/바깥쪽 중 그릴 위치를 설정합니다. **Fill** 영역에서는 색상 외에 **모서리 반경(Corner Radius %)** 도 지정합니다. 0은 각진 모서리이고, 100은 짧은 변 쪽이 완전히 둥근 알약 모양입니다.
- **Line** - 직선입니다. **From**과 **To**를 각각 X %/Y %로 지정하고, 선의 두께·색상·위치를 설정합니다.
- **Circle** - 원 또는 호입니다. 채우기와 테두리 외에도 **시작 각도(Start Angle)** 와 **끝 각도(End Angle)** 로 원의 일부만 그릴 수 있습니다. **부채꼴 그리기(Draw Slice)** 를 선택하면 가는 호 대신 부채꼴로 채웁니다.

그 밖의 요소

나머지 요소는 각각 별도 페이지에서 설명합니다.

- 텍스트 요소 - 글꼴, 크기, 정렬
- 이미지 요소 - 이미지 업로드와 이미지 라이브러리 사용
- 게이지 요소 - 값을 표시하는 막대, 링, 미터
- 그룹과 참조 - 요소 정리와 재사용
- 캔버스와 버튼 표시 정보 - 상단 표시줄, 상태 아이콘, 빈 버튼

2.6 텍스트 요소

텍스트(Text) 요소는 버튼에 글자를 표시합니다. 고정된 이름, 변수, 표현식의 결과를 사용할 수 있습니다.
대부분의 버튼에는 텍스트 요소가 하나 이상 있습니다.

텍스트 요소에는 공통 **그리기(Drawing)** 및 **위치와 크기(Position & Size)** 영역 외에 **내용(Content)** 영역이 있습니다.
공통 설정은 버튼 스타일을 참고하십시오.

표시할 텍스트

버튼 텍스트(Button text string) 필드에 표시할 내용을 입력합니다. 필드 옆의 전환 버튼으로 다음 두 가지 입력 방식을 선택할 수 있습니다.

- **변수 모드(Variables mode)** - 일반 텍스트에 $(connection:variable) 구문으로 변수를 넣습니다.
  예를 들어 Now on: $(internal:time_hms) 또는 Volume $(myMixer:fader1)dB를 입력합니다.
- **표현식 모드(Expression mode)** - 필드 전체를 표현식으로 처리합니다. 조건 판단이나 형식 지정이 필요할 때 적합합니다.
  예를 들어 $(myMixer:muted) ? 'MUTED' : 'LIVE'를 사용할 수 있습니다.

줄을 강제로 바꾸려면 Camera\n1처럼 줄바꿈할 위치에 \n을 입력합니다.

글꼴

글꼴(Font) 드롭다운에서 서체를 선택합니다.

- **기본(Default)** - Companion의 표준 가변 폭 글꼴입니다. 대부분의 이름 표시에 적합합니다.
- **고정 폭(Monospace)** - 모든 문자의 폭이 같은 글꼴입니다. 값이 바뀌어도 정렬을 유지해야 하는 숫자,
  타이머, 표 등에 유용합니다.

텍스트 스타일도 설정할 수 있습니다.

- **굵기(Weight)** - **보통(Normal)** 또는 **굵게(Bold)** 를 선택합니다.
- **스타일(Styles)** - **기울임(Italic)**, **밑줄(Underline)**, **취소선(Strikethrough)** 을 원하는 조합으로 켤 수 있습니다.
  표현식으로 전환하면 ['italic', 'underline']처럼 적용할 스타일의 목록을 반환해야 하며, 적용하지 않으려면 []를 반환합니다.
  예를 들어 $(custom:standby) ? ['italic'] : []는 사용자 정의 변수가 참일 때 텍스트를 기울임으로 표시합니다.

텍스트 크기

이전 Companion에서는 하나의 글자 크기 설정이 두 가지 역할을 했지만, 5.0부터는 다음 두 설정으로 나뉩니다.

- **텍스트 크기(Text Size)** - 요소 높이에 대한 백분율로 텍스트 크기를 지정합니다. 원하는 기본 크기로 설정하십시오.
- **영역에 맞게 축소(Shrink to fit)** - 선택하면 텍스트가 너무 길어 영역에 들어가지 않을 때 지정한 크기보다 작게 줄일 수 있습니다. 영역에 들어가는 가장 큰 크기를 사용하며, 최소 크기는 요소 높이의 10분의 1입니다. 선택하지 않으면 텍스트가 잘리더라도 지정한 크기를 유지합니다.

트랙 이름이나 타이머처럼 길이가 달라지는 텍스트에는 읽기 편한 **Text Size**를 지정하고 **Shrink to fit**을 켜는 방법이 일반적입니다. 그러면 짧은 값은 큼직하게 표시되고 긴 값도 영역 안에 들어갑니다.

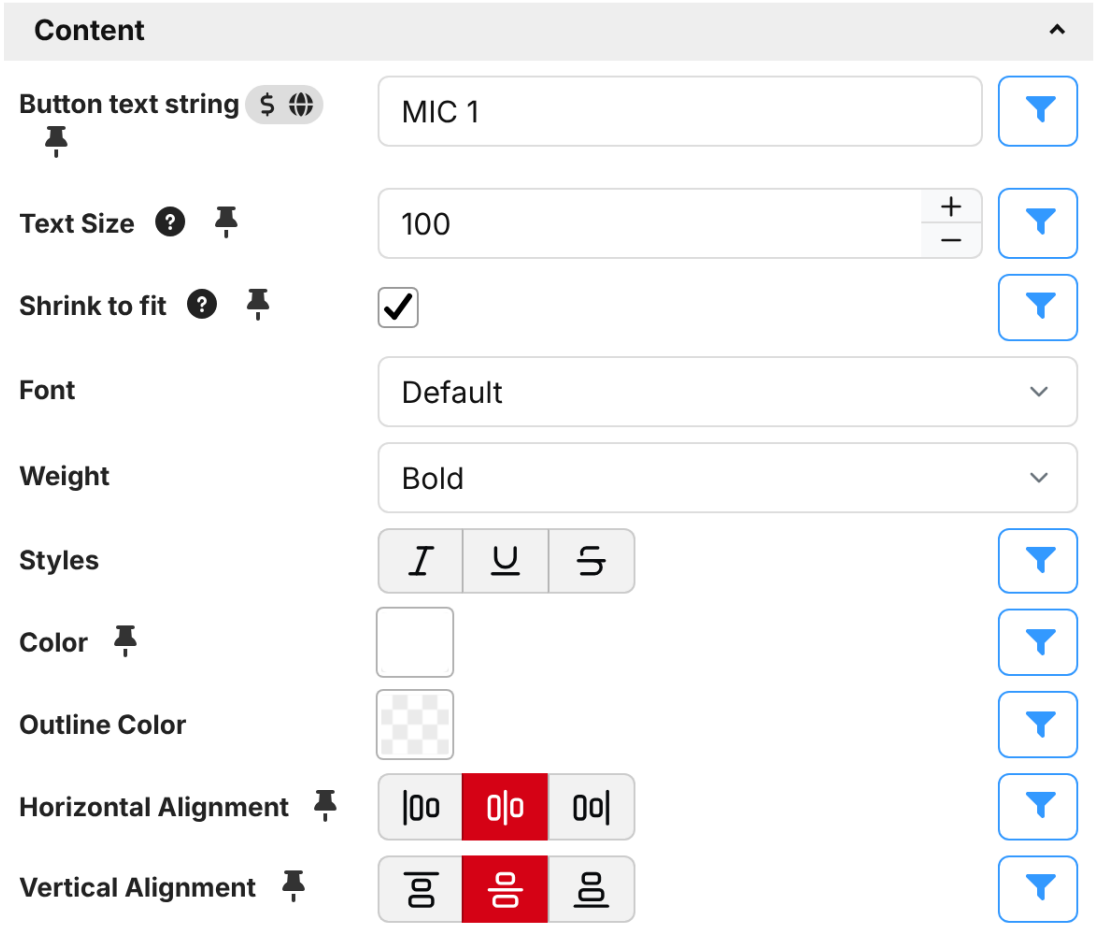


그림 2-27. 텍스트 요소 속성

도움말

크기의 기준은 버튼 전체가 아닌 요소 입니다. 따라서 모서리에 작은 텍스트 요소를 만들면 글자 크기도 그 요소에 맞춰집니다. **Position & Size**에서 요소 크기를 바꾸면 텍스트 크기도 함께 바뀝니다.

색상과 정렬

- **색상(Color)** - 텍스트의 색상입니다.
- **윤곽선 색상(Outline Color)** - 각 글자 주변에 선택적으로 윤곽선을 표시합니다. 예를 들어 검은색처럼 대비가 있는 윤곽선을 사용하면 복잡한 이미지 위에서도 글자를 쉽게 읽을 수 있습니다.

- **가로 정렬(Horizontal Alignment)** - 왼쪽, 가운데, 오른쪽 중에서 선택합니다.
- **세로 정렬(Vertical Alignment)** - 위쪽, 가운데, 아래쪽 중에서 선택합니다.

2.7 이미지 요소

이미지(Image) 요소는 아이콘, 로고, 카메라 썸네일 등 원하는 그림을 버튼에 표시합니다. 이미지 요소 위에 텍스트 요소를 올려 설명을 붙일 수도 있습니다.

이미지 요소에는 공통 **그리기(Drawing)** 및 **위치와 크기(Position & Size)** 영역 외에 **내용(Content)** 영역이 있습니다. 공통 설정은 버튼 스타일을 참고하십시오.

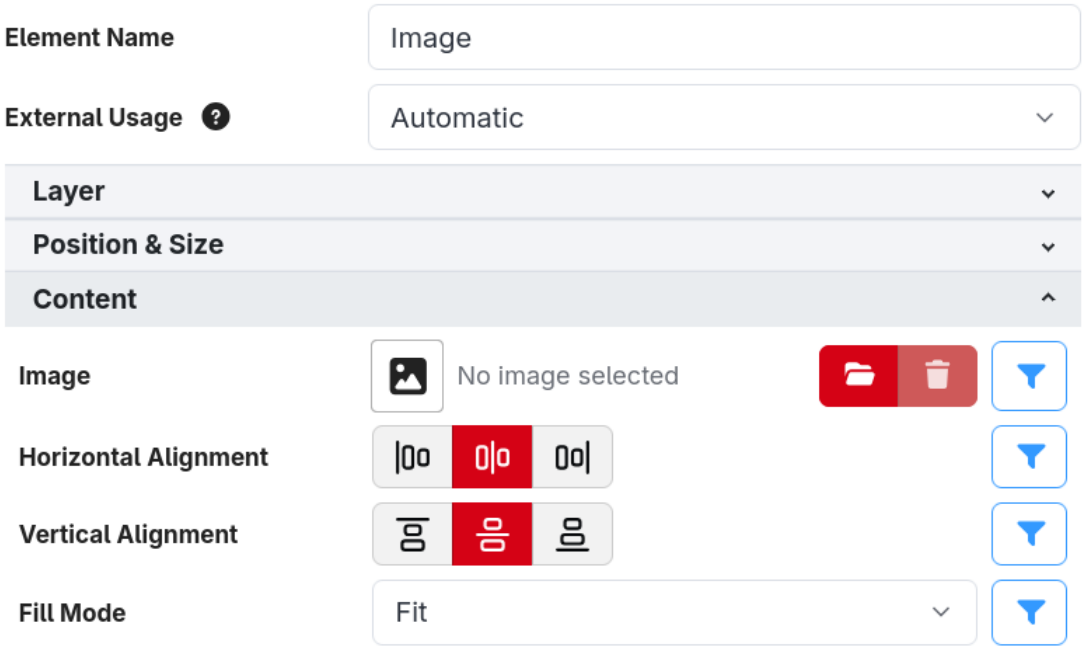


그림 2-28. 이미지 요소 속성

이미지 선택하기

이미지(Image) 필드에서 다음 두 가지 방법으로 이미지를 선택할 수 있습니다.

1. **직접 업로드하기. 이미지 선택(Select image)** 을 클릭하고 컴퓨터에서 파일을 선택합니다.
   이미지는 해당 버튼 안에 저장됩니다.
2. **이미지 라이브러리 참조하기.** 이미지 라이브러리에 올려 둔 이미지를 선택합니다. 같은 이미지를 여러 버튼에서 재사용할 때 권장하는 방법입니다. 라이브러리에서 이미지를 한 번 수정하면 이를 참조하는 모든 버튼도 함께 바뀝니다.

라이브러리 이미지는 $(image:name) 구문으로 이름을 지정해 참조하므로, 원한다면 표현식에 참조를 직접 입력하거나 붙여 넣을 수도 있습니다.

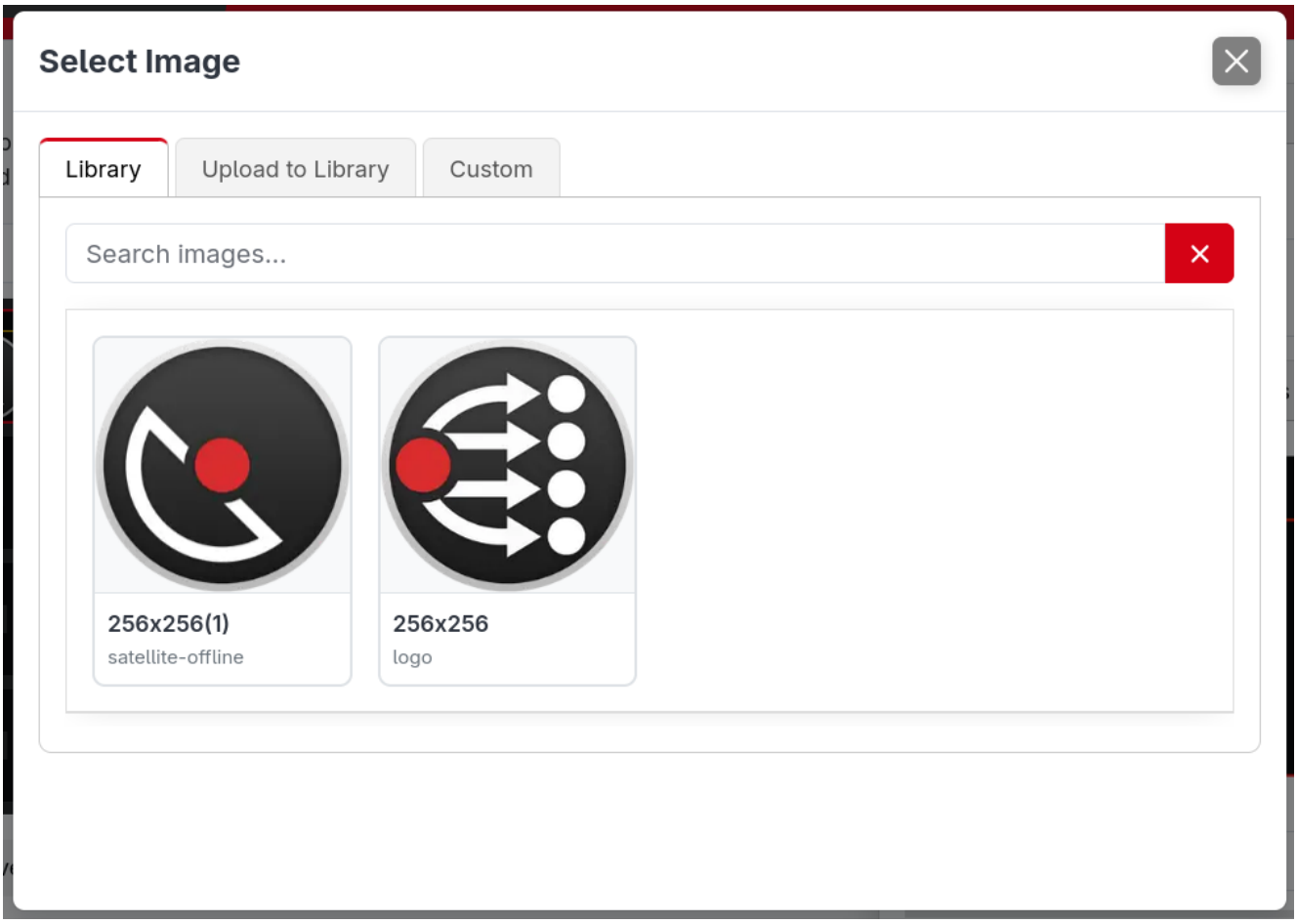


그림 2-29. 이미지 선택 창

크기 맞춤과 위치

- **채우기 모드(Fill Mode)** 는 이미지를 요소 크기에 맞추는 방식을 정합니다.
  - **맞춤(Fit)** - 이미지 전체가 요소 안에 보이도록 크기를 조정합니다. 기본값입니다.
  - **채우기(Fill)** - 요소 전체를 덮도록 크기를 조정하며, 넘치는 부분은 자릅니다.
  - **자르기(Crop)** - 크기를 조정하지 않고 이미지를 요소 크기에 맞게 자릅니다.
- **가로 정렬(Horizontal Alignment) / 세로 정렬(Vertical Alignment)** - 이미지가 요소를 완전히 채우지 않을 때 요소 안에서의 위치를 정합니다.

2.8 게이지 요소

게이지(Gauge) 요소는 값이 커질수록 채워지는 막대나 링으로 숫자를 표시합니다. 오디오 레벨, 페이더 위치, 클립 재생 진행률, 배터리 잔량, 타이머처럼 숫자를 읽기보다 한눈에 상태를 확인하는 편이 편리한 값에 적합합니다.

게이지 요소에는 공통 **그리기(Drawing)** 및 **위치와 크기(Position & Size)** 영역 외에도 전용 설정 영역이 있습니다.
공통 설정은 버튼 스타일을 참고하십시오.

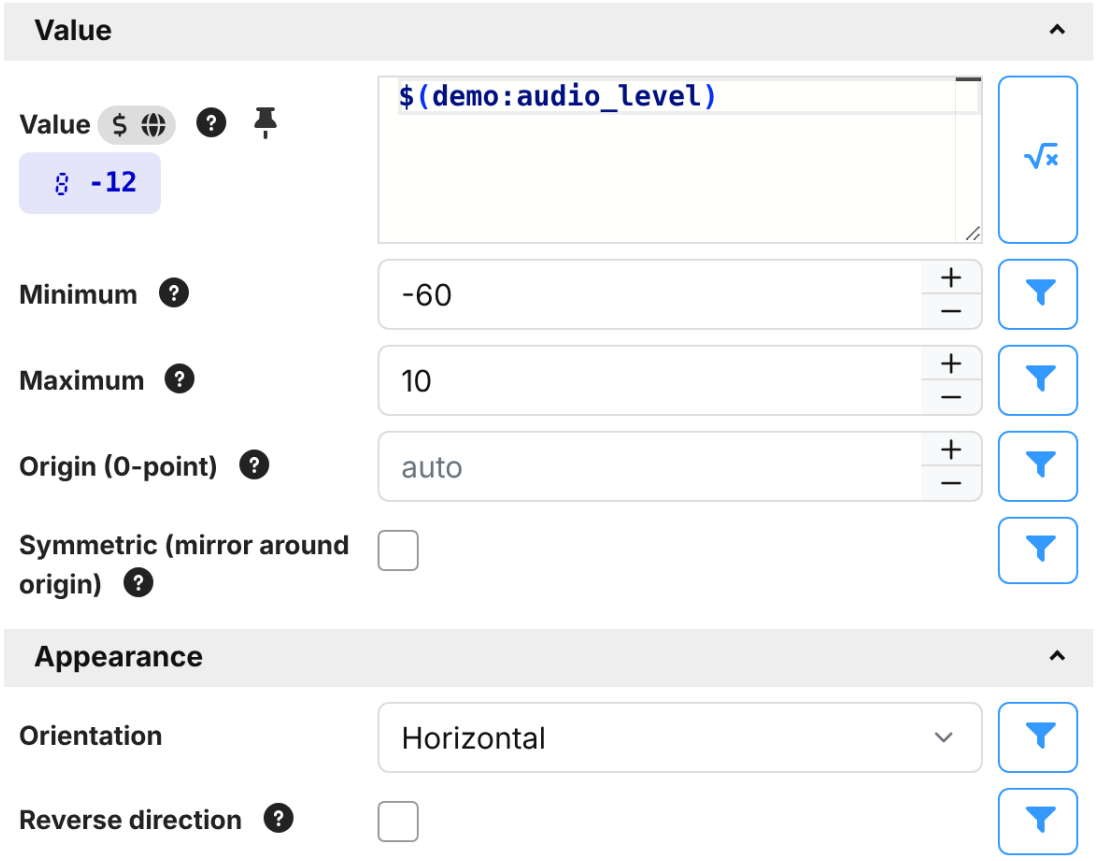


그림 2-30. 게이지 요소 속성

값

- **값(Value)** - 표시할 값입니다. 대부분 $(mixer:fader1_level) 같은 변수나 표현식을 사용합니다.
- **최솟값(Minimum)** / **최댓값(Maximum)** - 게이지의 범위입니다. 값이 최솟값이면 비어 있고, 최댓값이면 가득 찹니다. 범위를 벗어난 값도 각각 비거나 가득 찬 상태로 표시합니다.
- **기준점(Origin (0-point))** - 채우기가 시작되는 값입니다. 비워 두면 **자동(auto)** 으로 설정되어 최솟값부터 채워집니다. 범위 중간의 값으로 지정하면 그 지점을 기준으로 양쪽 방향으로 채울 수 있습니다. 예를 들어 -100~100 범위의 팬 제어에서 기준점을 0으로 설정할 수 있습니다.
- **대칭(Symmetric (mirror around origin))** - 켜면 값이 커질수록 기준점에서 양쪽으로 대칭으로 채워집니다. 스테레오 폭 같은 값을 표시하기에 적합합니다.

모양

- **방향(Orientation)** - **가로(Horizontal)**, **세로(Vertical)**, **링(Ring)** 중에서 선택합니다.

- **방향 반전(Reverse direction)** - 반대쪽부터 채웁니다. 가로는 오른쪽에서 왼쪽, 세로는 위에서 아래, 링은 반시계 방향입니다.

원형 스타일(Circular styling) 영역은 **Ring** 방향에만 적용됩니다.

- **시작 각도(Start angle) / 끝 각도(End angle)** - 링의 시작점과 끝점으로, 위쪽을 기준으로 시계 방향의 각도를 지정합니다. 두 지점 사이에 간격을 두면 원 대신 호를 그립니다. 예를 들어 225~135로 설정하면 아래쪽이 열린 다이얼 모양의 호가 됩니다.
- **링 두께(Ring width (%))** - 요소의 짧은 변에 대한 백분율로 링의 두께를 지정합니다.
- **끝 둥글게(Rounded ends)** - 링에서 채워진 부분의 끝을 둥글게 만듭니다.

채우기

- **채우기 표시(Show fill)** - 게이지의 채워진 부분을 표시합니다.
- **채우기 두께(Fill width (%))** - 사용 가능한 공간에 대한 채우기 두께입니다. 트랙보다 얇게 설정하면 넓은 트랙 위에 가는 채움 막대가 표시됩니다.
- **색상 지점(Color stops)** - 채우기에 사용할 색상을 지정합니다. 각 지점에는 해당 색상이 시작되는 **값(Value)** 과 **색상(Color)** 이 있습니다. 예를 들어 레벨 미터를 최솟값부터 녹색, -12부터 노란색, 0부터 빨간색으로 표시할 수 있습니다. **다음 색상까지 그라데이션(Gradient to next)** 을 선택하면 색상이 갑자기 바뀌지 않고 다음 지점의 색상으로 부드럽게 이어집니다. 색상에 부분 투명도를 적용할 수도 있습니다. **지점 추가(Add stop)** 로 새 지점을 추가합니다.
- **여러 색상으로 채우기(Multi-color fill)** - 켜면 일반적인 레벨 미터처럼 각 구간이 해당 색상으로 표시됩니다. 끄면 채워진 부분 전체가 현재 값에 해당하는 한 가지 색상으로 표시됩니다.

마커

마커는 현재 값의 위치에 표시되는 선입니다.

- **마커 표시(Show marker)** - 마커를 표시합니다.
- **색상(Color)** - 마커의 색상입니다.
- **두께(Width (%))** - 마커의 두께입니다.

트랙: 배경

트랙은 게이지에서 채워지지 않은 부분입니다.

- **스타일(Style)** - **투명(Transparent)** 은 트랙을 비워 두어 게이지 뒤의 내용이 보이게 합니다. **어둡게(Dimmed (darker))** 는 채우기 색상을 어둡게 표시해 게이지의 전체 범위를 확인할 수 있게 합니다.
- **밝기(Amount (%))** - 트랙을 어둡게 표시할 때 남겨 둘 색상의 밝기입니다. 0은 검은색이며, 100은 채우기 색상과 같습니다.
- **트랙 두께(Track width (%))** - 사용 가능한 공간에 대한 트랙의 두께입니다.

제어 장치의 LED에 게이지 표시하기

Stream Deck Studio의 다이얼 주변 링처럼 제어 장치의 표시등에도 게이지를 표시할 수 있습니다.
표시등에는 화면의 게이지와 같은 색상이 나타납니다. 해당 기능을 지원하는 제어 장치 모듈이 필요합니다.

이런 조작부에 게이지가 있는 버튼을 배치하면 Companion은 버튼의 첫 번째 게이지를 표시등에 표시합니다.
게이지가 여러 개이고 다른 게이지가 표시된다면, 표시할 게이지의 **외부 사용(External Usage)** 을 **Gauge**로 설정하십시오.
External Usage가 **Gauge** 또는 **자동(Automatic)** 이 아닌 게이지는 표시등에 사용되지 않습니다.

표시등에서는 다음 사항이 다릅니다.

- 마커는 표시되지 않습니다.
- 링 모양 표시등의 **Ring** 게이지에는 **Start angle**과 **End angle**이 적용되며, 호 바깥쪽의 표시등은 꺼집니다.
  일렬로 된 표시등이나 다른 방향의 게이지에서는 전체 표시등에 걸쳐 값을 표시합니다.
- 투명 색상은 검은색 배경 위에 표시됩니다.

2.9 그룹과 참조

그룹(Group) 과 **참조(Reference)** 요소를 사용하면 디자인을 정리하고 재사용할 수 있습니다.

그룹

그룹(Group) 은 다른 요소를 담는 컨테이너입니다. 요소를 그룹 안으로 끌어 놓거나 그룹을 선택한 상태에서 새 요소를 추가하면 하위 요소가 됩니다. 요소 목록에서는 들여쓰기되어 표시됩니다.

그룹을 사용하면 다음과 같은 이점이 있습니다.

- 그룹 전체를 한 번에 이동하거나 크기·회전을 변경할 수 있으며, 하위 요소도 함께 움직입니다.
- 그룹의 **사용(Enabled)** 필드로 여러 요소를 한꺼번에 표시하거나 숨길 수 있습니다.
- 복잡한 버튼의 요소를 깔끔하게 정리할 수 있습니다.

빈 그룹은 요소를 추가할 때까지 목록에 "Empty group"으로 표시됩니다.

정사각형 비율

그룹의 **옵션(Options)** 영역에는 **정사각형 비율(Square Aspect Ratio)** 전환 버튼이 있습니다. 켜면 하위 요소의 좌표 영역이 그룹의 짧은 변을 기준으로 한 정사각형으로 제한되며, 그룹 중앙에 배치됩니다.

이 옵션은 버튼이 정사각형이 아닐 때 화살표 같은 도형이 늘어나는 문제를 해결합니다. **Square Aspect Ratio**를 켠 그룹 안에 화살표를 넣으면 세로로 길거나 가로로 넓은 제어 장치에서도 화살표 모양이 유지됩니다. Companion의 기본 페이지 이동 버튼도 이 방식으로 화살표 모양을 유지합니다.

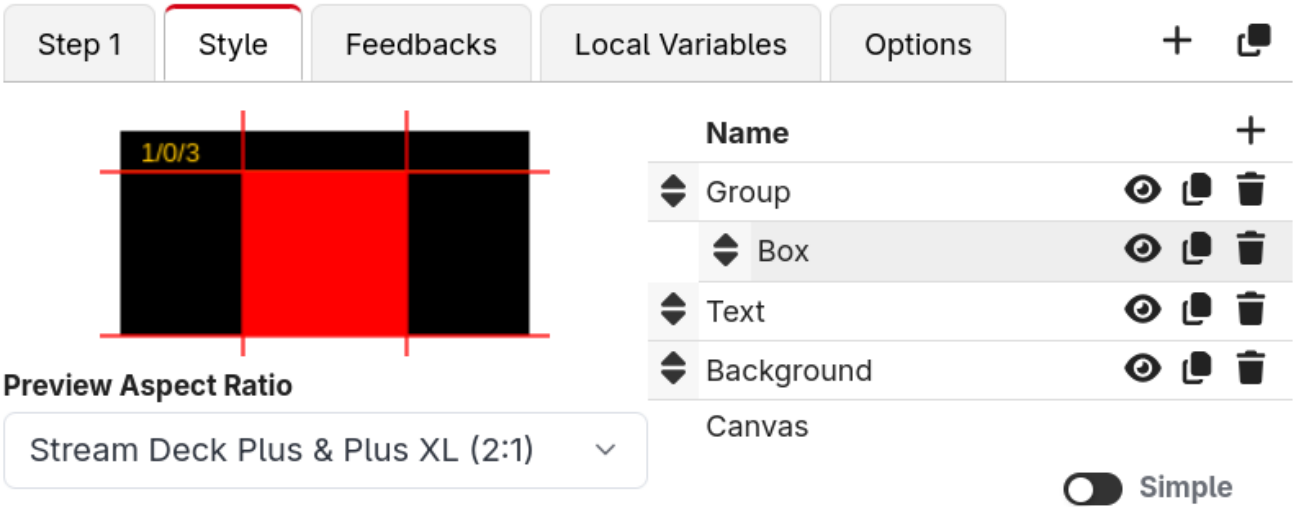


그림 2-31. 정사각형 비율을 적용한 그룹

참조

참조(Reference) 요소는 다른 버튼에 그려진 그래픽을 그대로 표시합니다. 원본 버튼에 그려지는 내용이 이 요소에도 똑같이 나타납니다.

도움말

참조 요소는 다른 버튼의 모양만 복사하며, 현재 버튼의 다른 요소 사이 어디에나 배치할 수 있습니다.
누름 동작까지 포함해 버튼 전체가 다른 버튼처럼 작동하게 하려면 버튼 참조를 사용하십시오.

디자인을 재사용할 때 특히 유용합니다. 원하는 모습으로 버튼 하나를 만든 뒤 다른 버튼의 참조 요소가 이 버튼을 가리키도록 설정하면, 원본을 수정할 때 모두 함께 바뀝니다.

요소의 **원본(Source)** 영역에서 **위치(Location)** 를 참조할 버튼의 page/row/column 형식으로 설정합니다.
예를 들어 1/0/0은 페이지 1, 행 0, 열 0입니다. 표현식도 사용할 수 있으므로 상황에 따라 참조 대상을 바꿀 수 있습니다.
예를 들어 $(this:page)/0/0은 항상 현재 페이지의 왼쪽 위 버튼을 참조합니다.

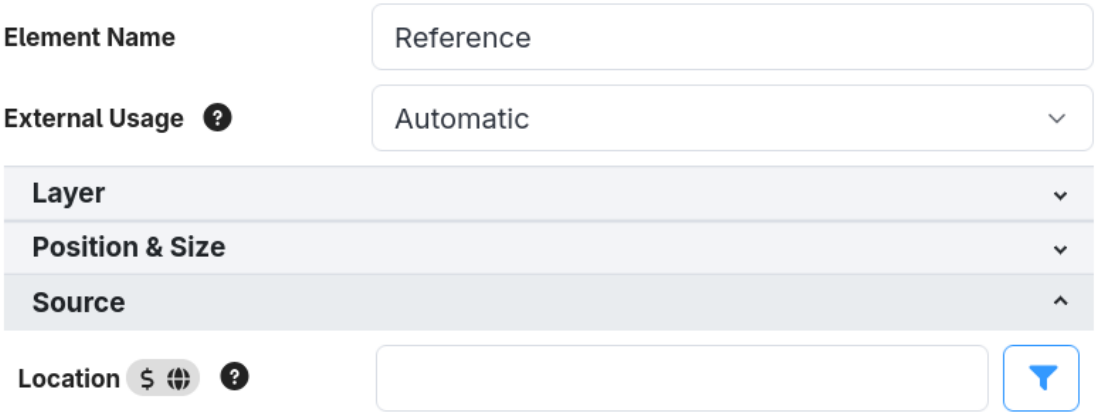


그림 2-32. 참조 요소

2.10 캔버스와 버튼 표시 정보

버튼의 요소는 모두 특수한 **캔버스(Canvas)** 레이어 위에 쌓입니다. 요소 목록 아래의 **Canvas** 버튼을 클릭하면 편집할 수 있습니다. 캔버스는 디자인이 그려지는 바탕이며, 버튼 주변의 상단 표시줄과 상태 아이콘을 제어합니다.

참고

캔버스와 **배경(Background)** 요소를 혼동하지 마십시오. 배경은 새 버튼에 기본으로 추가되는 일반 상자(Box) 요소로, 캔버스 바로 위에 있습니다. 다른 요소처럼 색상을 바꾸거나 이동·삭제할 수 있습니다. 캔버스는 그 아래에 고정된 레이어입니다.

장식 표시: 상단 표시줄

Canvas를 선택하면 **장식(Decoration)** 드롭다운이 나타납니다. 여기서 버튼 위쪽의 표시줄을 설정합니다.

- **기본 설정 따르기(Follow default)** - 설정의 공통 설정을 따릅니다. 기본 옵션이며, 한 곳에서 모든 버튼의 상단 표시줄을 켜거나 끌 수 있습니다.
- **상단 표시줄(Top bar)** - 상단 표시줄을 항상 표시합니다. 버튼의 위치(페이지/행/열)를 표시하며, 여러 단계가 있는 버튼에서는 현재 단계도 표시합니다.
- **누를 때 테두리 표시(Border when pressed)** - 상단 표시줄을 숨기고, 버튼을 누르는 동안 색상 테두리를 표시해 눌린 상태를 확인할 수 있게 합니다.
- **없음(None)** - 상단 표시줄과 누름 상태 테두리를 모두 표시하지 않습니다.

도움말

상단 표시줄은 버튼 위쪽의 공간을 차지합니다. 이미지나 텍스트를 가장자리까지 배치하려면 여기 또는 공통 설정에서 상단 표시줄을 끄십시오. 버튼의 전체 영역을 사용할 수 있습니다.

버튼을 누르고 있는 동안에는 **누름 상태**가 표시됩니다. **Top bar** 모드에서는 상단 표시줄이 색으로 채워지고, **Border when pressed** 모드에서는 색상 테두리가 나타납니다. 보통 잠깐만 표시되지만, 버튼을 누른 뒤 해제하지 않으면 계속 표시됩니다. 예를 들어 **Button Trigger Press** 액션을 실행하고 대응하는 **Button Trigger Release** 액션을 실행하지 않은 경우입니다.

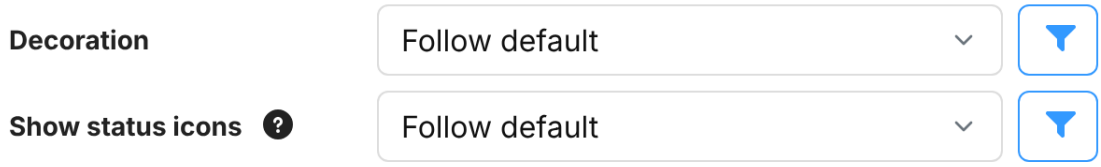


그림 2-33. 캔버스 레이어 옵션

상태 아이콘

상태 아이콘 표시(Show status icons) 드롭다운에서 오른쪽 위에 나타나는 작은 아이콘을 설정합니다.

- **기본 설정 따르기(Follow default)** - 공통 설정을 따릅니다.
- **모두 표시(Show all)** - 상태 아이콘을 항상 표시합니다.
- **없음(None)** - 상태 아이콘을 표시하지 않습니다.

이 아이콘은 주의가 필요한 상태를 한눈에 알려 주며, 해당 상태가 되면 자동으로 나타납니다.

표시	의미
빨간색 삼각형	이 버튼의 액션에서 사용하는 연결 중 하나 이상이 **오류** 상태입니다.
주황색 삼각형	이 버튼에서 사용하는 연결이 **경고** 상태입니다.
녹색 재생 화살표	이 버튼의 액션이 **실행 중**이거나 **대기 중**입니다. 예를 들어 **internal: Wait** 액션이 포함된 지연 실행이나 **순차(Sequential)** 모드의 액션 그룹이 실행 중인 경우입니다.



그림 2-34. 상태 표시 아이콘

빈 버튼

액션과 요소가 없는 버튼은 흐린 회색으로 표시됩니다. 버튼이 비어 있어 사용할 수 있다는 뜻이며, 요소나 액션을 추가하면 정상적인 모습으로 표시됩니다.

2.11 액션

액션은 버튼을 누르거나 놓을 때, 또는 외부에서 버튼을 실행할 때 수행할 동작을 정의합니다.

하나의 버튼에 여러 액션을 연결할 수 있으며, 서로 다른 모듈의 액션을 함께 사용할 수도 있습니다.
대부분의 액션에는 동작을 조정할 수 있는 옵션이 있습니다.

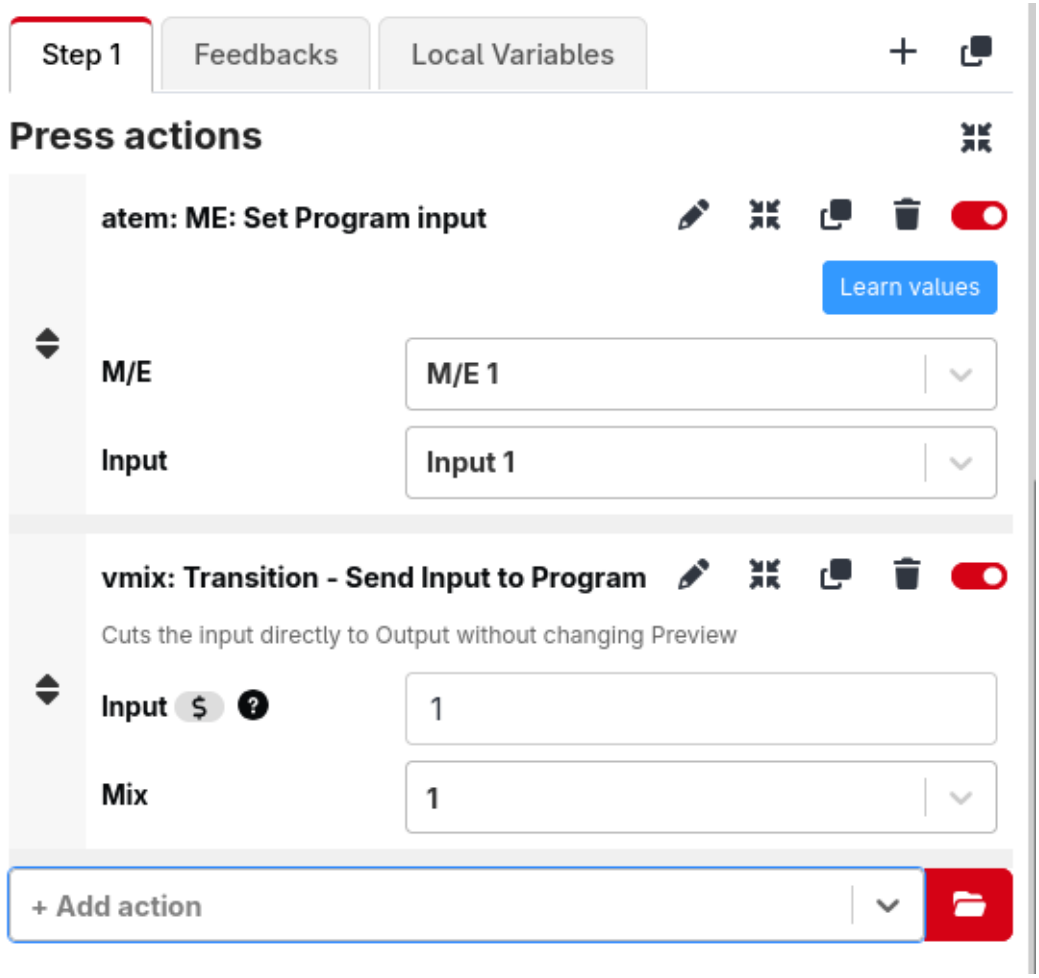


그림 2-35. 버튼 액션

버튼에 액션을 추가하려면 아래 검색란에서 검색하거나 폴더 버튼을 클릭합니다. 폴더 버튼을 누르면 사용 가능한 모든 액션을 자세히 살펴볼 수 있습니다.

Companion에서는 모듈이 제공하는 액션뿐 아니라 다양한 내장 액션도 사용할 수 있습니다.

각 버튼에는 실행할 액션을 담는 여러 그룹이 있습니다.

누름 액션(Press actions) 은 버튼을 누르거나 외부에서 실행할 때 수행됩니다.

해제 액션(Release actions) 은 버튼에서 손을 뗄 때 수행됩니다.

길게 누르는 동작을 위해 시간별 그룹을 추가할 수도 있습니다. **누름 시간 그룹 추가(Add duration group)** 버튼으로 그룹을 만든 뒤, 기준 시간과 실행 시점을 설정합니다. 손을 뗄 때 실행할지, 계속 누르고 있는 동안 실행할지 선택할 수 있습니다.

누름 시간 그룹을 추가하면 **Release actions**가 **짧게 누른 뒤 해제 액션(Short release actions)** 으로 바뀝니다.
이 액션은 첫 번째 시간 그룹의 기준 시간에 도달하기 전에 손을 뗄 때만 실행됩니다.

Companion 3.5부터 기존 지연 설정은 **internal: Wait** 액션으로 대체되었습니다. 이 액션은 목록에서 뒤따르는 모든 액션을 지연시킵니다. 최상위 액션은 아래의 **동시(Concurrent)** 모드로 실행됩니다.

실행 방식 변경하기

복잡한 동작을 구성할 수 있도록 **internal: Action Group** 액션도 제공됩니다. 이 액션은 다른 액션을 담는 그룹 역할을 하며, 또 다른 **Action Group**도 포함할 수 있습니다.

그룹의 실행 모드는 다음과 같습니다.

- **상위 설정 따르기(Inherit)** - 상위 그룹의 실행 모드를 따릅니다.
- **동시(Concurrent)** - 이전 Companion에서 **절대 지연(Absolute Delays)** 을 켠 것과 같습니다. 액션이 병렬로 실행되며, **internal: Wait** 액션을 만날 때마다 실행 구간이 나뉩니다.
- **순차(Sequential)** - 앞 액션이 끝날 때까지 기다린 뒤 다음 액션을 실행합니다. 모든 모듈이 액션의 완료 여부를 Companion에 알려 주는 것은 아니지만, 많은 모듈이 지원합니다. 이를 이용하면 짧은 지연 시간을 일일이 넣지 않고도 정해진 순서대로 수행해야 하는 복잡한 동작을 구성할 수 있습니다.

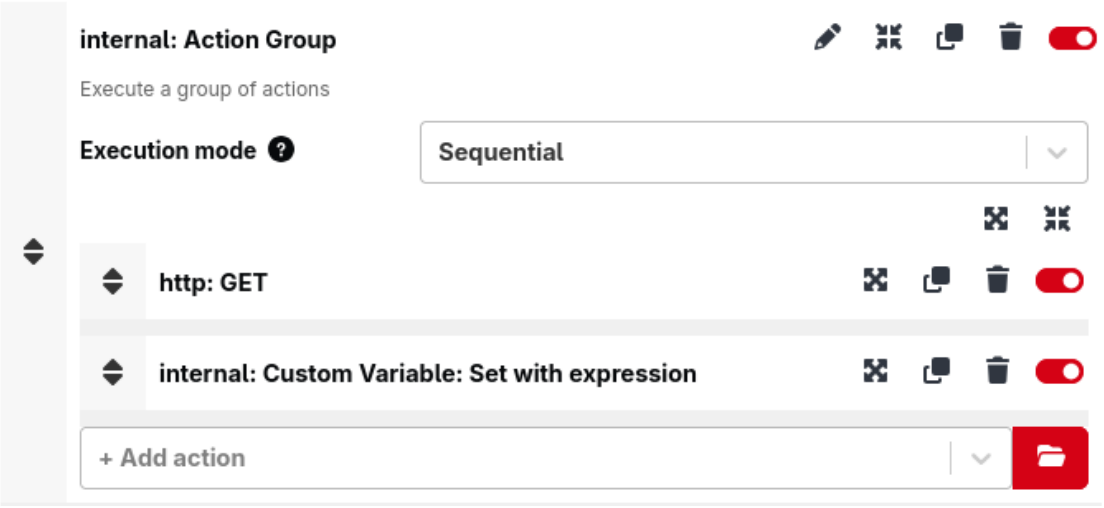


그림 2-36. 순차 실행 액션 그룹

조건부 액션과 반복

Logic: If statement 액션은 **When True**의 조건이 모두 참이면 **Then** 액션을, 그렇지 않으면 **Else** 액션을 실행합니다. **Logic: While loop** 액션은 **While True**의 모든 조건이 참인 동안 액션을 반복합니다.

조건에는 피드백을 사용합니다. 조건에 포함된 내장 피드백은 이전에 계산한 값을 사용하는 대신 액션을 실행하는 순간 확인합니다. 따라서 다음과 같이 동작합니다.

- 바로 앞에서 실행한 액션이 바꾼 값까지 포함해 현재 변수 값을 확인합니다. **While loop**는 반복할 때마다 최신 값을 확인하므로, 반복문 안의 액션으로 반복 종료 조건에 쓰이는 변수를 바꿀 수 있습니다.
- 버튼을 누른 제어 장치의 $(this:surface_id)나 회전 액션의 $(this:delta)처럼 액션을 실행하게 한 상황 정보를 사용할 수 있습니다.

연결에서 제공하는 피드백도 조건으로 사용할 수 있지만, 이 경우에는 연결에서 마지막으로 보고한 값을 사용합니다.

도움말

While loop 안에는 **Wait** 액션을 넣으십시오. 대기가 없으면 초당 수백 회까지 가능한 한 빠르게 반복되어 CPU 사용량이 늘고 장치에 지나치게 많은 명령이 전송될 수 있습니다. 보통 반복 사이에 100ms 정도의 짧은 대기만 넣어도 충분합니다.

액션 결과 저장하기

장치에서 값을 가져오거나 쿼리를 실행하는 일부 모듈 액션은 결과를 반환합니다. 이런 액션에는 **액션 결과 저장(Store Action Result)** 옵션이 표시되며, 결과를 버리지 않고 다른 곳에서 사용하도록 저장할 수 있습니다.

결과는 다음 변수에 저장할 수 있습니다.

- **로컬 변수(Local variable)** - 버튼 위치와 값을 저장할 로컬 변수 이름을 선택합니다.
  현재 버튼의 위치는 $(this:page)/$(this:row)/$(this:column)처럼 지정합니다.
- **페이지 변수(Page variable)** - 페이지와 값을 저장할 페이지 변수의 이름을 선택합니다.
- **사용자 정의 변수(Custom variable)** - 기존 사용자 정의 변수를 선택합니다. 아직 없다면 새로 만들 수도 있습니다.

결과가 필요하지 않으면 **<discard>** 를 선택합니다.

저장한 값은 일반 변수처럼 버튼 텍스트, 피드백, 다른 액션에서 참조할 수 있습니다. 이 옵션은 실제로 결과를 반환하는 액션에만 나타나며, 대부분의 액션은 결과를 반환하지 않습니다.

2.12 피드백

참고 5.0의 변경 사항

Companion 5.0에서 피드백 방식이 크게 바뀌었습니다. 버튼이 여러 그래픽 요소를 쌓아 구성하는 방식으로 바뀌면서, 불리언 피드백은 고정된 버튼 스타일을 설정하는 대신 **해당 요소의 속성을 덮어써서** 모양을 변경합니다.
이전 버전을 사용했다면 아래의 피드백으로 버튼 모양 바꾸기를 읽어 보십시오.

피드백(Feedback) 은 현재 상황에 따라 버튼 모양을 자동으로 바꾸는 기능입니다. 별도 조작 없이도 카메라가 라이브 상태일 때 버튼을 빨간색으로 바꾸거나, 믹서 채널이 음소거되면 다른 문구를 표시하거나, 레이어가 송출 중일 때 버튼을 밝힐 수 있습니다.

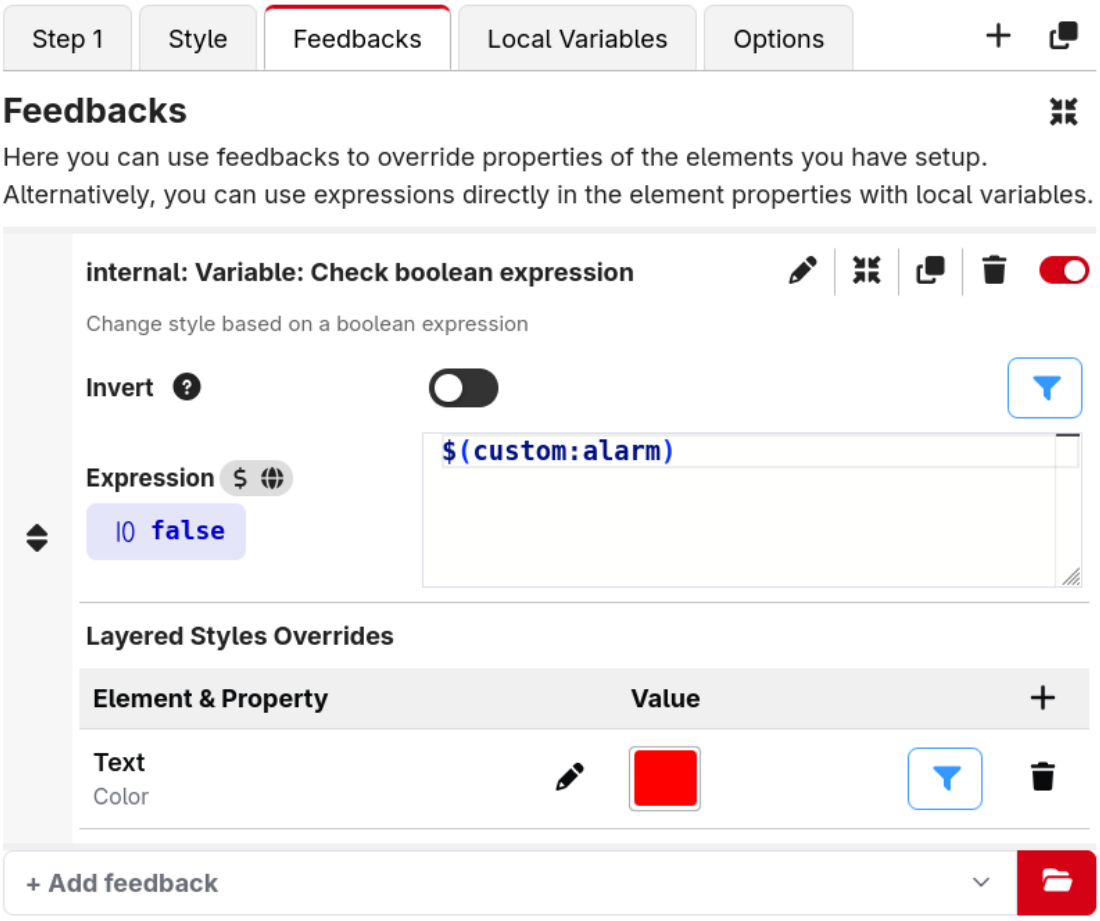


그림 2-37. 피드백 목록

일부 피드백은 모듈이 장치 상태를 보고하는 연결에서 제공됩니다. Companion에는 "변수가 특정 값이면 스타일 변경"과 같은 일반적인 논리를 위한 내장 피드백도 있습니다.

피드백 없이 모양 바꾸기

버튼이 변화에 반응하도록 만드는 방법은 피드백만 있는 것이 아닙니다. **그래픽 요소의 모든 속성에는 고정값 대신 표현식을 사용할 수 있습니다.** 표현식에서 변수를 참조하고 계산할 수 있으며, 사용 중인 변수가 바뀌면 요소가 자동으로 다시 그려집니다.

따라서 피드백 없이도 다양한 동적 스타일을 만들 수 있습니다. 예를 들면 다음과 같습니다.

- 텍스트 요소의 내용을 변수 값으로 바로 표시합니다.
- 텍스트 요소의 **색상(Color)** 을 $(custom:armed)에 따라 빨간색 또는 녹색을 반환하는 표현식으로 설정합니다.
- 요소의 **사용(Enabled)** 필드로 표시 여부를 제어합니다. 예를 들어 blink(500)으로 깜박이게 합니다.

피드백은 **연결에서 보고하는 상태**에 따라 모양을 바꾸거나, 하나의 켜짐/꺼짐 조건으로 **여러 속성 변경을 한꺼번에 적용하거나 해제**할 때 유용합니다. 각 속성에 표현식을 쓰는 것보다 변경 규칙을 설정하는 방식이 편할 때도 적합합니다.
두 방식을 함께 사용할 수 있으므로 만들려는 버튼을 더 쉽게 구성할 수 있는 방법을 선택하십시오.

피드백으로 버튼 모양 바꾸기

Companion 5.0의 버튼은 여러 그래픽 요소를 쌓아 구성하므로, 피드백은 조건이 참인 동안 **요소의 속성을 덮어쓰는 방식**으로 작동합니다.

각 피드백에는 변경할 요소 속성을 나열하는 **스타일 덮어쓰기(Style Overrides)** 표가 있습니다.
항목은 다음과 같이 추가합니다.

1. 표 머리글의 **+** (Add override, 덮어쓰기 추가)를 클릭합니다. **요소 속성 추가(Add Element Properties)** 창이 열립니다.
2. 영향을 줄 **요소**를 선택합니다. 예를 들어 **텍스트(Text)**, **이미지(Image)**, **배경(Background)** 을 선택합니다.
3. 해당 요소에서 덮어쓸 **속성**을 선택합니다. 텍스트의 **색상(Color)**, **텍스트 크기(Text Size)**, 내용 자체 또는 배경의 **색상(Color)** 등이 가능합니다. 여러 속성을 선택한 뒤 **속성 추가(Add Properties)** 를 클릭합니다.
4. **값(Value)** 열에서 적용할 값을 설정합니다. 피드백이 활성화된 동안 이 값이 요소의 평소 설정을 대신합니다.

행의 연필 아이콘으로 대상 요소와 속성을 바꾸고, 휴지통 아이콘으로 항목을 삭제합니다. 덮어쓸 항목이 없는 피드백은 아무 효과가 없으며, 해당 안내가 표시됩니다.

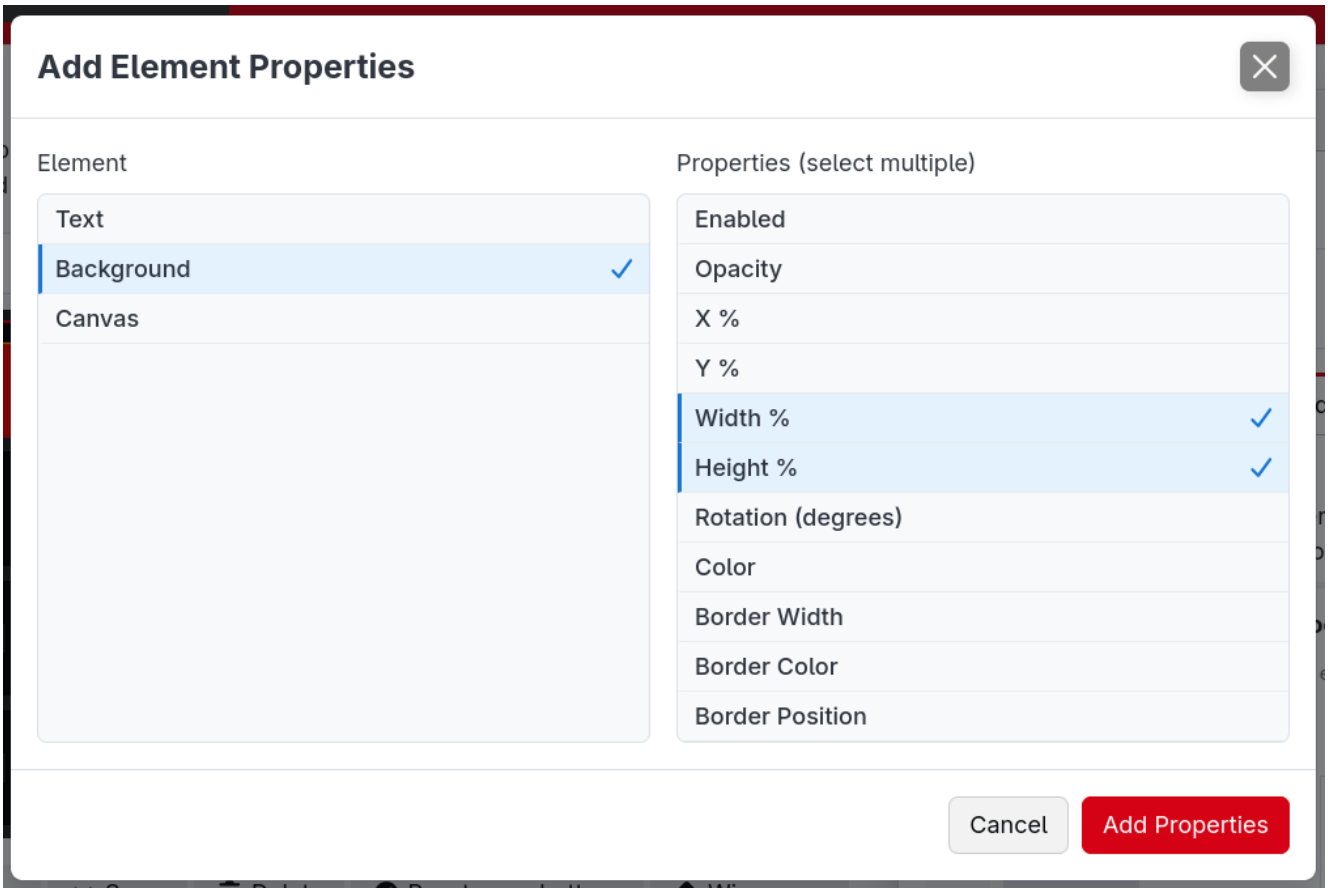


그림 2-38. 요소 속성 추가

하나의 피드백에 여러 요소와 속성의 덮어쓰기를 추가할 수 있습니다. 정렬 핸들을 드래그해 피드백 순서도 바꿀 수 있습니다. 피드백은 위에서 아래로 적용되므로 두 피드백이 같은 속성을 변경하면 목록의 **아래쪽** 피드백이 우선합니다.

참고 Companion 4.x와의 차이

이전 버전과 우선순위가 반대입니다. 4.x에서는 목록의 위쪽 피드백이 우선했지만, 5.0에서는 아래쪽, 즉 나중에 적용되는 피드백이 우선합니다.

간단한 예

사용자 정의 변수 alarm이 참일 때 버튼 텍스트를 빨간색으로 바꾸려면 다음과 같이 설정합니다.

1. 표현식을 검사하는 내장 피드백을 추가하고 표현식을 $(custom:alarm)으로 설정합니다.
2. 해당 피드백의 **Style Overrides** 표에서 **+**를 클릭한 뒤 **Text** 요소를 선택하고 **Color** 속성을 추가합니다.
3. 색상을 빨간색으로 설정합니다.

이제 alarm이 참이면 텍스트가 빨간색이 되고, 참이 아니면 원래 색상으로 돌아갑니다.

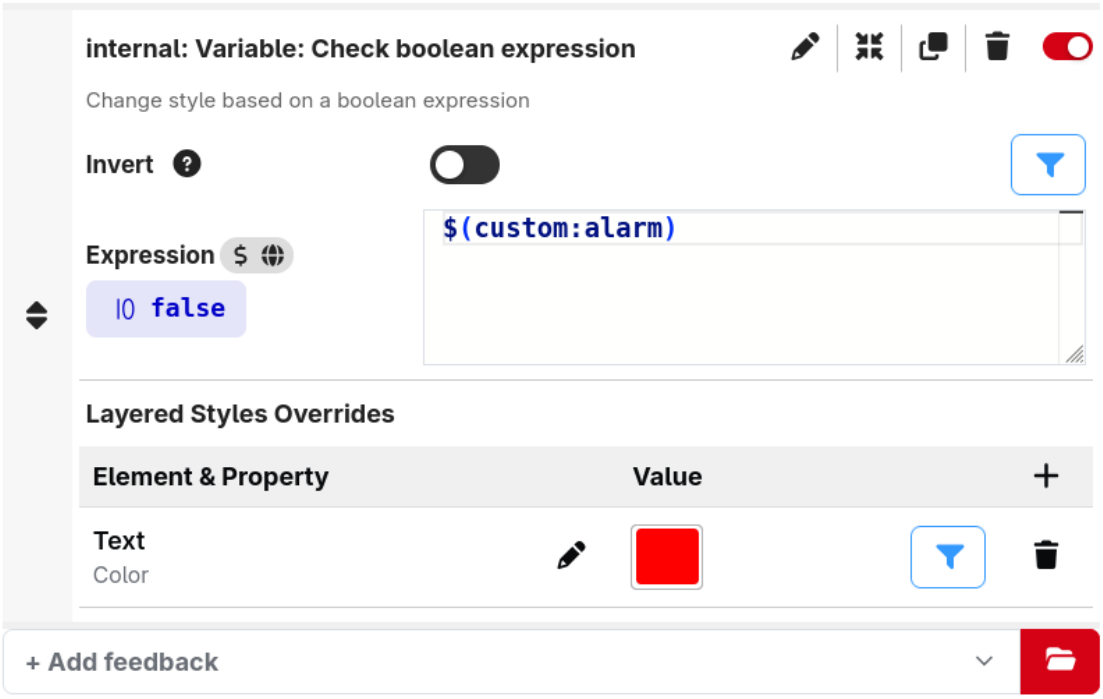


그림 2-39. 피드백 적용 예

도움말 Companion 4.x에서 업그레이드할 때

이전에는 "버튼 배경을 빨간색으로 변경"처럼 하나의 평면적인 버튼 스타일에 피드백을 적용했습니다.
개념은 같지만, 이제는 "**Background** 요소의 **Color**를 빨간색으로 설정"처럼 더 구체적으로 지정합니다.
단일 버튼 스타일에 항상 적용하는 대신, 피드백이 영향을 줄 요소 를 선택해야 합니다.

기존 설정을 다시 만들 필요는 없습니다. 저장된 설정을 열면 Companion이 기존 피드백을 기본 Text, Image, Background 요소를 대상으로 하도록 자동 변환하므로, 이전과 같은 모습이 유지됩니다.

피드백 유형

피드백에는 세 가지 유형이 있으며, 스타일 덮어쓰기를 적용하는 방식이 다릅니다. **Style Overrides** 제목 옆의 아이콘에 마우스를 올리면 유형을 확인할 수 있습니다.

- **불리언 피드백(Boolean feedbacks)** - 켜짐/꺼짐 방식입니다. 조건이 참일 때 설정한 덮어쓰기를 적용합니다. 위에서 설명한 표현식·변수 피드백이 대표적인 예입니다.
- **값 피드백(Value feedbacks)** - 레벨, 이름, 숫자 등의 값을 반환합니다. 아래의 값 피드백을 참고하십시오.
- **기존 스타일 피드백(Legacy style feedbacks)** - 일부 연결에서 제공하며, 장치 상태를 바탕으로 미터나 상태 색상 같은 전체 스타일을 반환합니다. 각 덮어쓰기 항목은 피드백이 반환하는 스타일 속성 하나를 골라 버튼 요소의 속성에 복사합니다.

값 피드백

값 피드백으로도 버튼 모양을 바꿀 수 있습니다. 값 피드백은 켜짐/꺼짐 상태 대신 값을 반환합니다. 연결에서 현재 페이더 레벨을 보고하는 피드백이나 내장 **표현식 계산(Evaluate Expression)** 피드백이 이에 해당합니다.

값 피드백에서는 각 덮어쓰기의 값이 항상 표현식이며, $(this:value)로 피드백의 현재 값을 가져올 수 있습니다.
새 덮어쓰기 항목의 기본 표현식은 $(this:value)로, 값을 속성에 그대로 복사합니다. 예를 들어 연결에서 받은 값을 요소의 텍스트로 표시할 수 있습니다. 값을 변환할 수도 있습니다. 다음은 레벨이 너무 높을 때 텍스트를 빨간색으로 바꾸는 예입니다.

```
$(this:value) > -6 ? 0xff4040 : 0xffffff
```

표현식이 아무 값도 반환하지 않거나(undefined) 실행에 실패하면 해당 속성은 요소 자체의 값을 유지합니다.

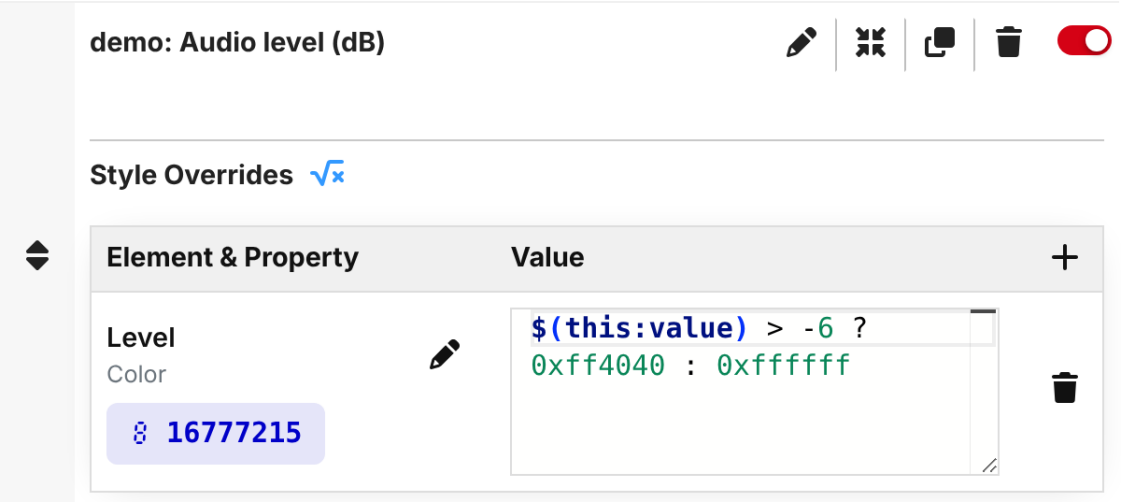


그림 2-40. 값 피드백의 스타일 덮어쓰기

요소 속성과 마찬가지로, Companion은 속성 이름 아래에 각 덮어쓰기 표현식의 현재 계산 결과를 표시합니다.

생성된 이미지

일부 모듈은 완성된 이미지를 그리는 피드백도 제공합니다. 일반적으로 복잡한 상태를 보기 좋게 정리해 표시하는 데 사용합니다.

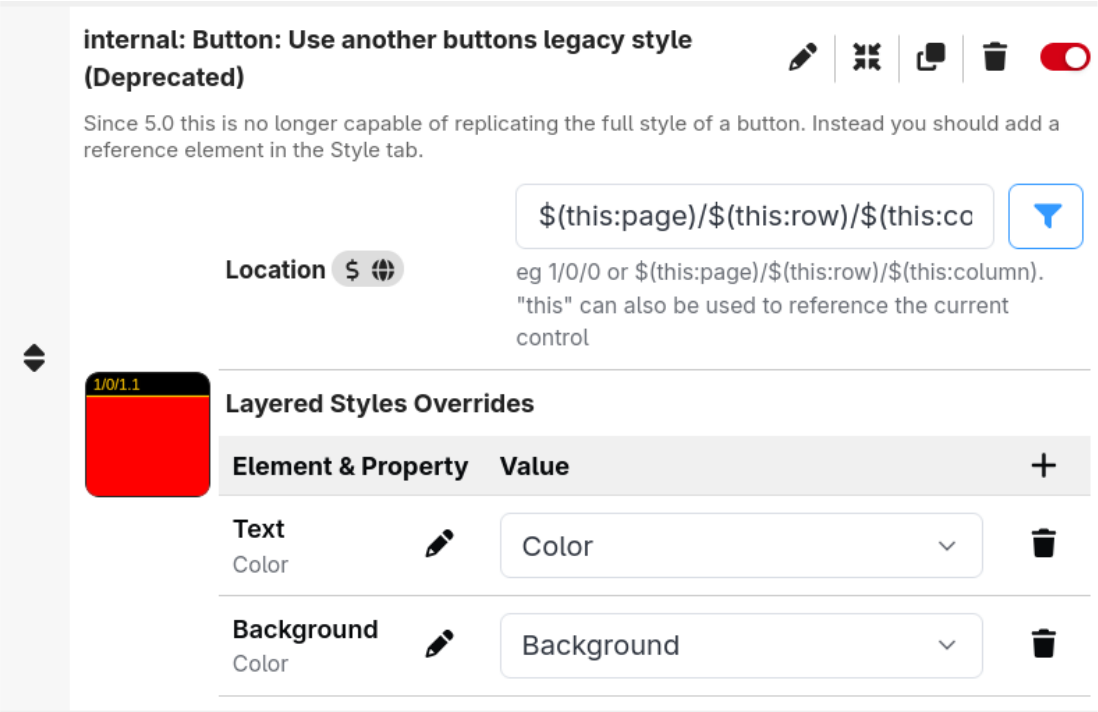


그림 2-41. 모듈 스타일 피드백

참고

일부 오래된 모듈의 피드백은 변경할 수 있는 항목이 제한적입니다. 원하는 구성을 만들기에 기능이 부족하다면 해당 모듈의 GitHub에 기능 개선을 요청하십시오.

2.13 단계

Companion 3.0부터 단계(Steps)를 사용해 다양한 복합 동작을 수행하는 버튼을 만들 수 있습니다.
이 기능은 이전 버전의 **래치(Latch)** 기능을 대체합니다.

단계 만들기

버튼에서 **단계 추가(Add step)** 탭을 클릭하면 두 번째 액션 탭이 생깁니다.

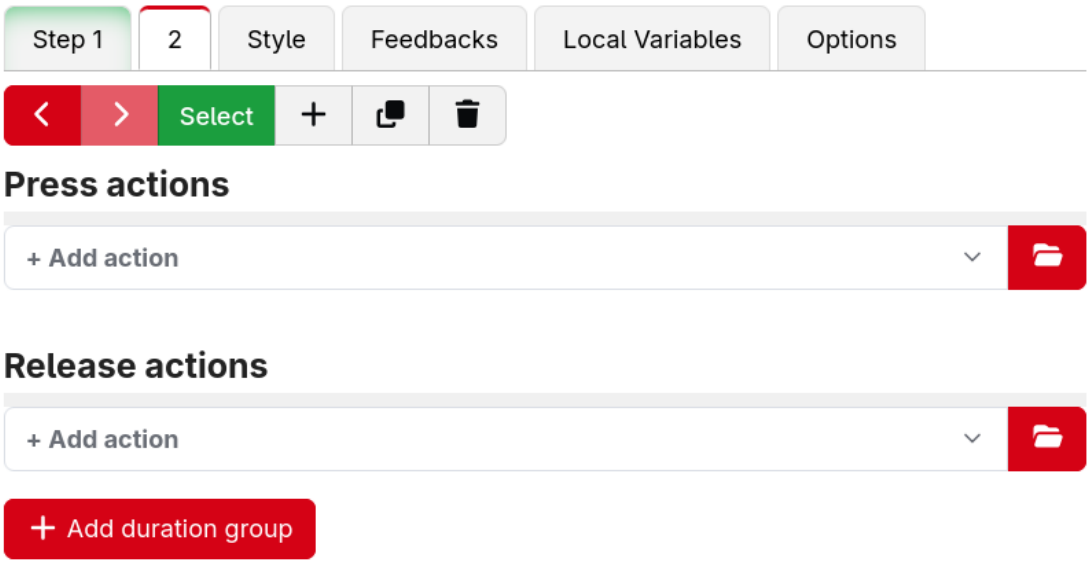


그림 2-42. 버튼의 2단계

이제 버튼을 처음 누르면 **1단계(Step 1)** 의 액션이 실행되고, 다시 누르면 **2단계(Step 2)** 의 액션이 실행됩니다.
이후에도 두 단계를 차례로 반복합니다.

사용 목적에 따라 버튼에 원하는 만큼 단계를 추가할 수 있습니다.

다음 단계로 넘어가는 방식

기본적으로 버튼에서 손을 뗄 때마다 다음 단계로 넘어갑니다. 내부 액션으로 버튼의 현재 단계를 바꾸거나 특정 단계로 설정할 수도 있습니다.

단계가 자동으로 넘어가지 않게 하려면 버튼 설정의 위쪽에 있는 **단계 진행(Step Progression)** 옵션을 변경합니다.
현재 단계 번호를 표현식으로 정하거나 내부 액션으로만 바뀌도록 선택할 수 있습니다.

이를 이용하면 한 버튼을 누르고 있는 동안 다른 버튼들이 2단계로 바뀌고, 손을 떼면 1단계로 돌아오는 Shift 레이어 같은 복합 동작을 구성할 수 있습니다.

예: 버튼을 한 번 누르면 프로젝터의 셔터를 닫고, 두 번째로 누르면 셔터를 열려면 다음과 같이 설정합니다.

- **1단계(Step 1)** 의 **누름 액션(Press actions)** 목록에 셔터를 닫는 액션을 추가합니다.
- **2단계(Step 2)** 의 **해제 액션(Release actions)** 목록에 셔터를 여는 액션을 추가합니다.

2.14 페이지

Companion v3.5부터 페이지 수를 원하는 만큼 설정할 수 있습니다.

새로 설치하면 페이지 1개로 시작하며, 필요한 만큼 추가할 수 있습니다. 몇 페이지만 사용하는 사용자도 있고 수백 페이지를 사용하는 사용자도 있습니다.

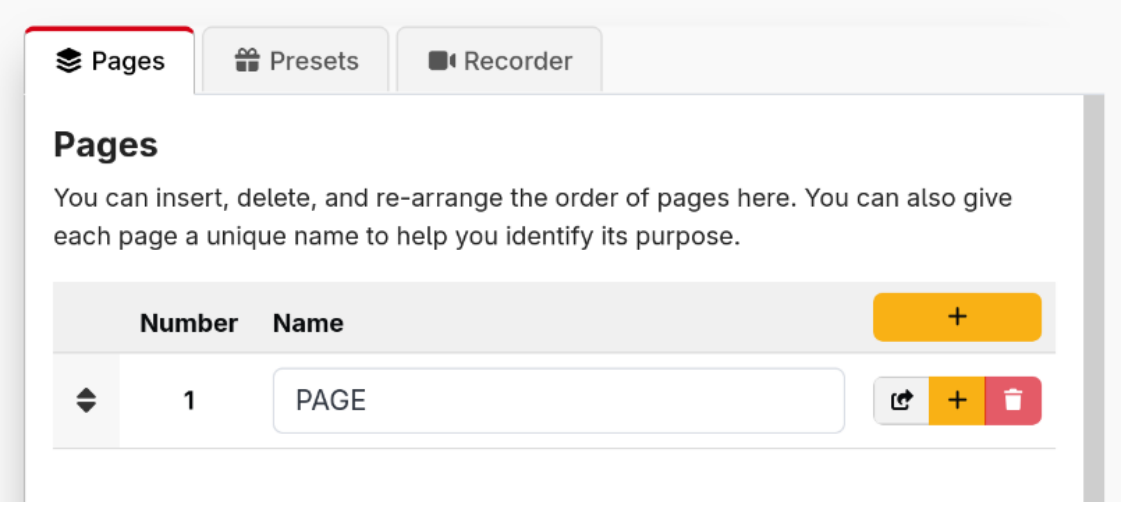


그림 2-43. 페이지 목록

도움말

페이지를 쉽게 구분할 수 있도록 각각 이름을 지정하는 것이 좋습니다.

도움말

제어 장치(Surface)를 여러 대 사용한다면 버튼을 배치하는 방법은 크게 두 가지입니다. 장치마다 별도의 페이지를 사용하거나, 여러 장치가 페이지를 공유하면서 그리드 오프셋 설정으로 장치마다 페이지의 다른 영역을 사용하도록 구성할 수 있습니다. 작업에 맞는 방법을 선택하십시오.

주의

페이지 순서를 바꾸거나 중간에 페이지를 삽입할 때는 주의하십시오. 페이지 번호를 가리키는 참조는 자동으로 갱신되지 않으므로 기존 버튼이 제대로 작동하지 않을 수 있습니다.

페이지 변수

각 페이지에는 자체 변수를 둘 수 있습니다. 버튼의 로컬 변수와 같은 방식으로 작동하지만, 해당 페이지의 모든 버튼이 공유합니다. 페이지의 버튼들이 제어할 카메라나 그 페이지를 사용할 사람의 이름처럼 페이지 전체에 적용할 값을 저장하기에 적합합니다.

페이지 변수는 버튼 그리드 옆의 **페이지 변수(Page Variables)** 탭에서 관리하며, 항상 그리드에 표시된 현재 페이지에 속합니다. 로컬 변수와 마찬가지로 **사용자 값(User Value)**, **표현식 계산(Evaluate Expression)**, **피드백(Feedback)** 유형을 사용할 수 있습니다.

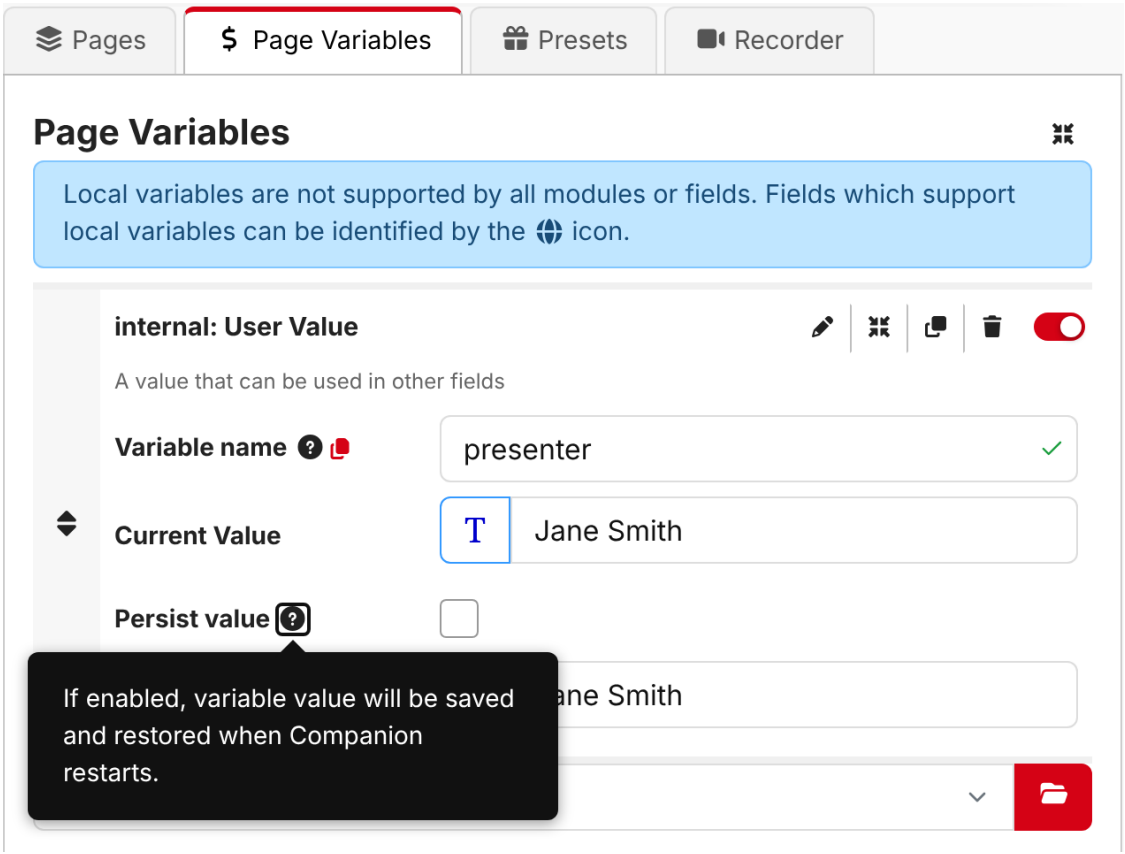


그림 2-44. 페이지 변수

해당 페이지의 버튼에서는 $(page:name)으로 페이지 변수를 사용합니다. 예를 들어 이름이 presenter인 페이지 변수는 $(page:presenter)로 참조합니다.

페이지 변수는 다른 페이지나 트리거를 포함해 어디에서든 내부 액션 **Page Variable: Set value**, **Page Variable: Reset to startup value**, **Page Variable: Write current value to startup value**로 변경할 수 있습니다.
각각 값 설정, 시작값으로 초기화, 현재 값을 시작값으로 저장하는 액션입니다. 이 액션에서 변경할 페이지를 지정하므로, 한 버튼으로 다른 페이지 전체가 제어할 대상을 선택할 수도 있습니다. 액션의 결과를 페이지 변수에 저장할 수도 있습니다.
액션 결과 저장하기를 참고하십시오.

참고

트리거는 특정 페이지에 속하지 않으므로 $(page:...) 변수를 직접 읽을 수 없습니다. 다만 위의 액션으로 페이지 변수를 변경할 수는 있습니다.

페이지를 내보내거나 가져올 때 페이지 변수도 함께 포함됩니다.

2.15 액션 기록기

액션 기록기(Action Recorder)는 연결된 장치를 직접 조작해 버튼의 액션 목록을 만드는 기능입니다.
예를 들어 스위처에서 입력을 전환하면 해당 조작을 기록하므로 액션을 하나씩 직접 추가할 필요가 없습니다.

참고

현재 액션 기록기를 지원하는 모듈은 매우 적습니다. 사용 중인 연결에서 이 기능을 지원하는지는 모듈에 따라 다르므로 모듈 설명서나 프리셋 안내를 확인하십시오.

액션 기록기 사용하기

1. 편집할 버튼을 열고 **액션(Actions)** 탭으로 이동합니다.
2. 액션 기록기 영역에서 **기록 시작(Start recording)** 을 클릭합니다.
3. 기록할 조작을 장치에서 수행합니다. 예를 들어 ATEM에서 소스를 전환합니다.
4. Companion이 조작을 감지하면 해당 액션이 기록 목록에 나타납니다.
5. 조작을 마쳤으면 **기록 중지(Stop recording)** 를 클릭합니다.
6. 기록된 액션을 확인하고 **저장(Save)** 을 클릭해 버튼에 적용하거나, **취소(Discard)** 를 클릭해 기록을 버립니다.

참고

- 모듈에서 지원하는 모든 액션을 기록할 수 있는 것은 아닙니다. 모듈이 Companion에 명시적으로 알리는 조작만 목록에 나타납니다.
- 기록기는 조작 당시의 상태를 기록합니다. 단계 사이에 지연이 있는 순서를 만들려면 기록한 뒤 internal: Wait 액션을 직접 추가해야 할 수 있습니다.

2.16 제어 장치

Companion의 제어 장치(Surface)는 키패드, 노브, 슬라이더 같은 물리적 입력 장치를 Companion의 버튼 격자에 연결합니다. 널리 쓰이는 제품으로 Elgato의 Stream Deck 시리즈와 XKeys가 있으며, 이들 장치의 제어 통합 기능은 Companion에 포함되어 있습니다. Companion 4.3부터는 여러 다른 제어 장치도 "독립형" 제어 장치 모듈로 지원합니다.
자세한 내용은 제어 장치 모듈 도움말을 참조하세요.

구성된 제어 장치

구성된 제어 장치는 Companion의 버튼을 실제 장치의 키·노브에 연결하는 관계를 정의합니다.
버튼 밝기 등 장치별 속성도 포함합니다.

중요

Companion이 새 제어 장치를 인식하려면 먼저 적절한 **제어 장치 통합** 모듈을 설치하고 활성화해야 합니다.

주의 Elgato 장치

목록에 Stream Deck이 나타나지 않으면 Elgato Stream Deck 애플리케이션을 닫고 Rescan 버튼을 클릭해야 할 수 있습니다.

구성된 제어 장치 목록에 장치가 없다면 **USB 다시 검색(Rescan USB)** 버튼을 누릅니다. 오른쪽 설정 패널에서 **새 USB 장치 감시(Watch for new USB Devices)**를 켜면 직접 다시 검색할 필요가 없습니다.

참고 Linux USB 권한

Linux에서 Companion이 USB 제어 장치를 사용하려면 udev 규칙을 설치해야 합니다.
필요한 규칙은 활성화한 제어 장치 모듈에 따라 달라집니다. 규칙을 업데이트해야 하면 Surfaces 페이지에 알림이 표시됩니다.
데스크톱 설치에서는 Companion이 규칙을 적용할 수 있으며, Headless 설치에서는 설치 스크립트가 설정합니다.
직접 설정하려면 Linux 다운로드에 포함된 README를 참조하세요.

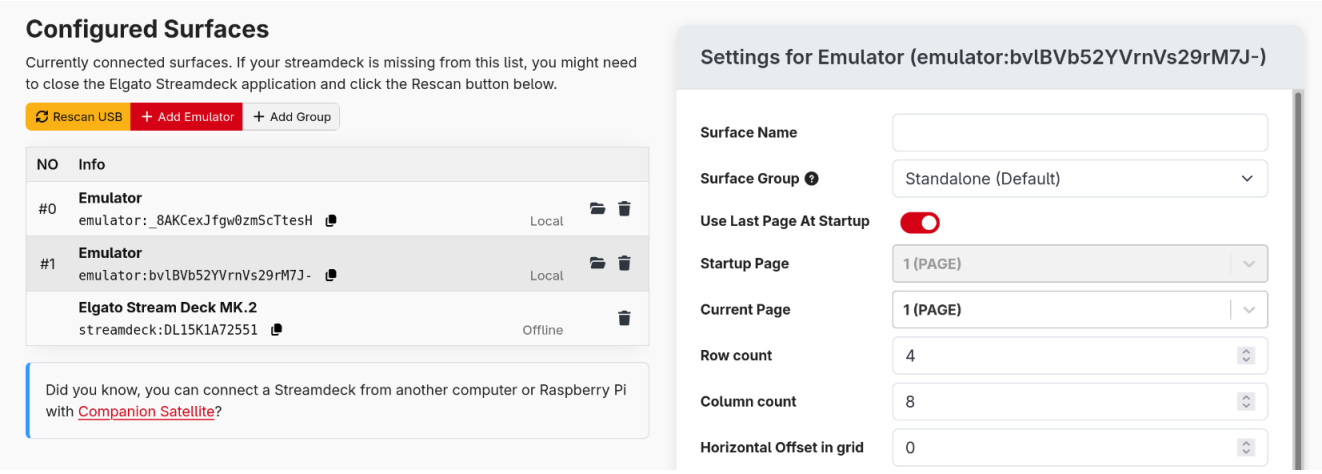


그림 2-45. 제어 장치

로컬 장치, Satellite를 통해 연결된 장치, 사용 가능한 에뮬레이터를 포함한 모든 현재 제어 장치를 확인할 수 있습니다.

제어 장치 활성화 및 비활성화

각 제어 장치의 설정에는 사용 여부를 전환하는 스위치가 있습니다. 끄면 Companion이 장치를 사용하지 않지만 설정은 그대로 유지됩니다. Elgato Stream Deck 앱 같은 다른 소프트웨어와 Companion을 함께 실행할 때 유용합니다. 특정 장치는 Companion 전용으로 두고 나머지는 다른 애플리케이션에서 관리할 수 있습니다.

도움말

이 스위치는 Satellite를 통해 연결된 제어 장치에는 적용되지 않습니다.

제어 장치 설정

표에서 제어 장치를 클릭하면 해당 장치의 설정이 열립니다.

정확한 설정 목록은 장치 유형에 따라 다르지만 일반적으로 다음 옵션을 제공합니다.

- **제어 장치 이름(Surface Name)**: 장치의 이름을 지정합니다. 화면에서 이 장치를 참조하는 모든 위치에 사용됩니다.
- **제어 장치 그룹(Surface Group)**: 장치를 제어 장치 그룹에 할당합니다.
- **시작 시 마지막 페이지 사용(Use Last Page At Startup)**: 시작할 때 마지막으로 사용한 페이지를 기억하여 사용할지, Startup Page 설정을 사용할지 선택합니다.
- **시작 페이지(Startup Page)**: 'Use Last Page At Startup'을 켜지 않았을 때 장치가 시작하며 표시할 페이지입니다.
- **격자의 가로 오프셋(Horizontal Offset in grid)**: 장치가 버튼 격자보다 작다면 격자 안에서 장치의 위치를 조정할 수 있습니다.
- **격자의 세로 오프셋(Vertical Offset in grid)**: 장치가 버튼 격자보다 작다면 격자 안에서 장치의 위치를 조정할 수 있습니다.
- **제어 장치 회전(Surface Rotation)**: 제조사가 의도한 방향과 다르게 장치를 설치하면 편리한 경우가 있습니다. 여기에서 방향을 보정합니다.
- **밝기(Brightness)**: 버튼의 밝기입니다.
- **버튼 회전(Button rotation)**: 제어 장치를 물리적으로 돌려 설치했다면 버튼 표시 방향을 맞춥니다.
- **PIN 잠금 제외(Never pin code lock)**: 이 장치를 PIN 코드 잠금에서 제외합니다.

페이지 접근 권한

v4.2부터 개별 제어 장치나 제어 장치 그룹에서 접근 가능한 페이지를 제한할 수 있습니다. 그룹 설정은 그룹 안의 장치들에 적용됩니다. 예시는 다음과 같습니다.

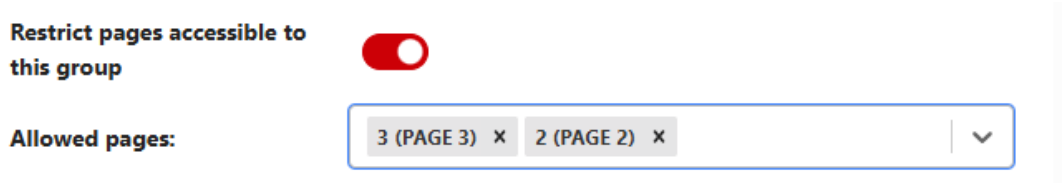


그림 2-46. 페이지 접근 제한 설정

하나의 현장 설정을 배포하면서 담당자의 역할에 따라 접근할 페이지를 지정할 수 있습니다. 각 장치나 그룹은 자신의 역할에 필요한 한 페이지 또는 몇 개의 페이지만 사용하도록 제한됩니다.

도움말

표준 페이지 버튼, 사용자 지정 액션, 지원 장치의 스와이프를 비롯한 다음/이전 페이지 이동은 여기에서 선택한 순서를 따릅니다. 순서를 바꾸려면 목록에서 항목을 삭제한 뒤 원하는 순서대로 다시 선택합니다.

제어 장치 그룹

여러 제어 장치를 그룹으로 연결하면 항상 같은 페이지를 표시하게 할 수 있습니다.
예를 들어 Stream Deck XL 두 대를 나란히 놓고 하나의 16x4 장치처럼 사용하려는 구성에 편리합니다.

+ 그룹 추가(+ Add Group) 버튼으로 새 제어 장치 그룹을 만듭니다. 그룹에 넣을 각 장치의 설정을 열고 제어 장치 그룹 드롭다운에서 새 그룹을 선택합니다.

이후 Companion의 다른 위치에서 제어 장치 페이지를 변경하는 액션에는 개별 장치 대신 그룹이 표시됩니다.
기존에 개별 장치를 참조하던 액션은 해당 장치가 속한 그룹에 적용됩니다.

그룹을 이용해 제어 장치의 ID를 일정하게 유지할 수도 있습니다.
Stream Deck을 빠르게 교체해야 하면서 액션에서는 특정 Stream Deck을 참조해야 하는 구성에 도움이 됩니다.
미리 그룹을 만들고 시스템을 설정한 뒤 나중에 그룹의 장치를 추가하거나 제거해도 버튼을 수정할 필요가 없습니다.

원격 제어 장치

일부 제어 장치는 네트워크를 통해 Companion에 연결할 수 있습니다. Companion Satellite나 Stream Deck Network Dock 등을 사용할 수 있습니다.

이 중 일부는 Companion이 제어 장치에 연결해야 하며 이 페이지에서 설정할 수 있습니다.

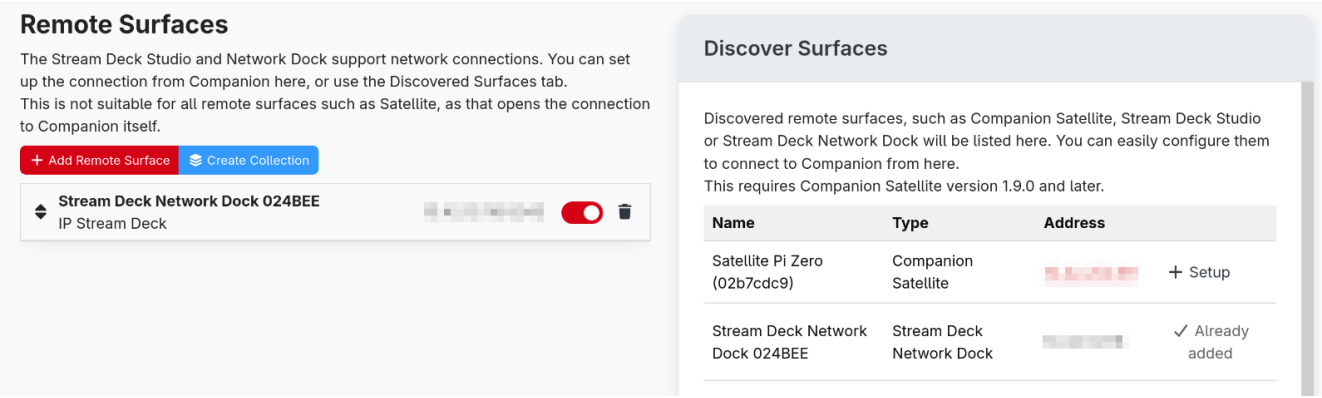


그림 2-47. 원격 제어 장치

왼쪽에는 설정된 연결이, 오른쪽에는 네트워크에서 검색되어 설정할 수 있는 제어 장치가 표시됩니다.

2.17 트리거

트리거(Triggers) 탭에서는 Companion 환경의 트리거를 추가·편집·제거할 수 있습니다.

트리거를 사용하면 간단한 자동 동작을 구성하여 여러 작업을 자동화할 수 있습니다.

새 트리거를 추가하려면 **트리거 추가(Add Trigger)** 버튼을 클릭하고 아래 정보를 입력합니다.

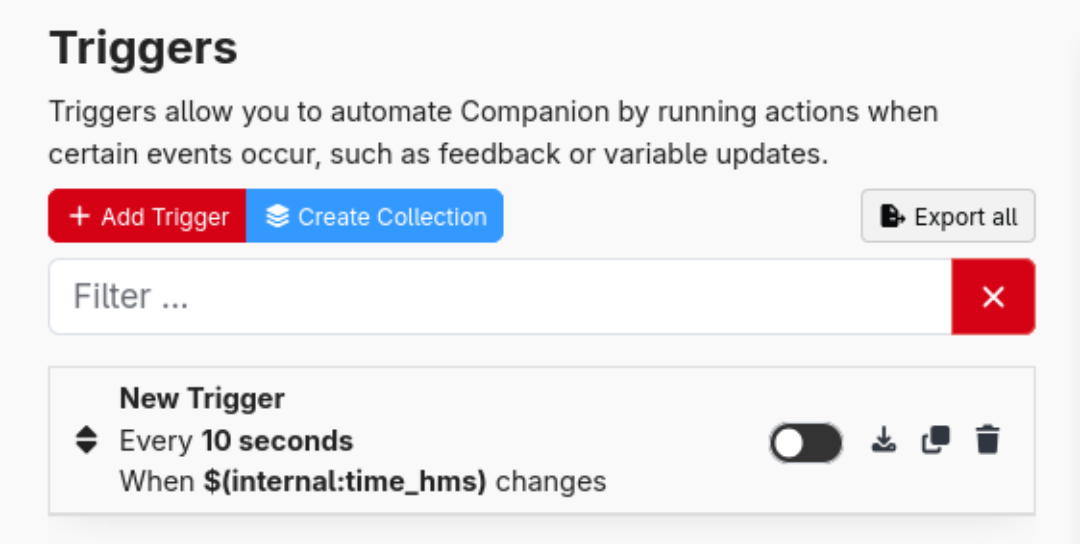


그림 2-48. 트리거

구성

추가한 트리거는 오른쪽 편집 패널에서 열 수 있습니다.

새 트리거는 준비를 마치기 전에 실행되지 않도록 처음에는 비활성화되어 있습니다.

먼저 이름을 지정하고 나머지 항목을 입력합니다.

트리거에는 설정할 구역이 3개 있습니다.

이벤트

트리거를 언제 실행할지 정의합니다.
등록한 이벤트 중 하나라도 발생하면 트리거가 실행됩니다.
자주 사용하는 이벤트는 **시간 간격(Time interval)**, **임의 시간 간격(At random time interval)**, **변수 변경 시(On variable change)**입니다.

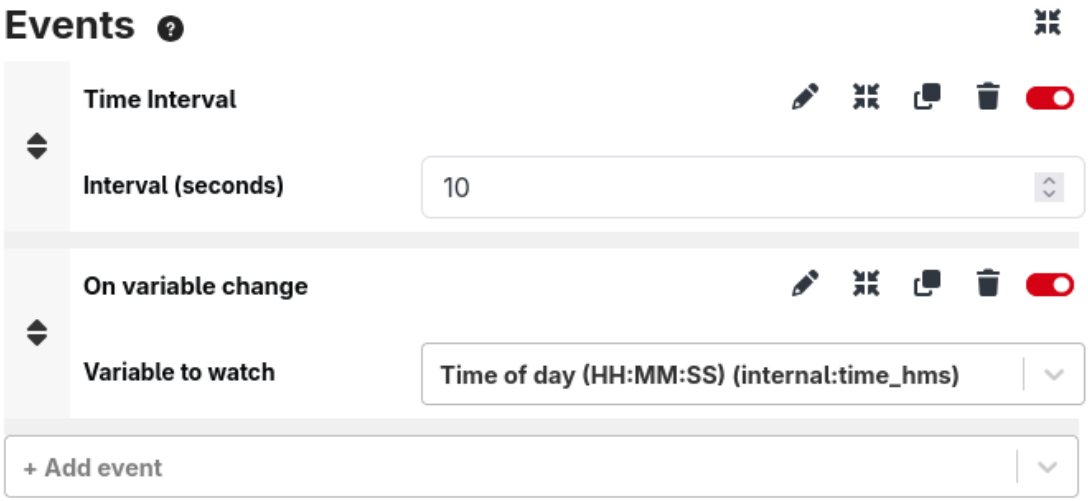


그림 2-49. 이벤트

조건

이벤트에 필터를 적용합니다.
예를 들어 **On variable change** 이벤트에서는 변수 값이 1일 때만 트리거를 실행하도록 조건을 설정할 수 있습니다.
버튼에서 사용할 수 있는 피드백 중 일부가 조건으로 제공됩니다.

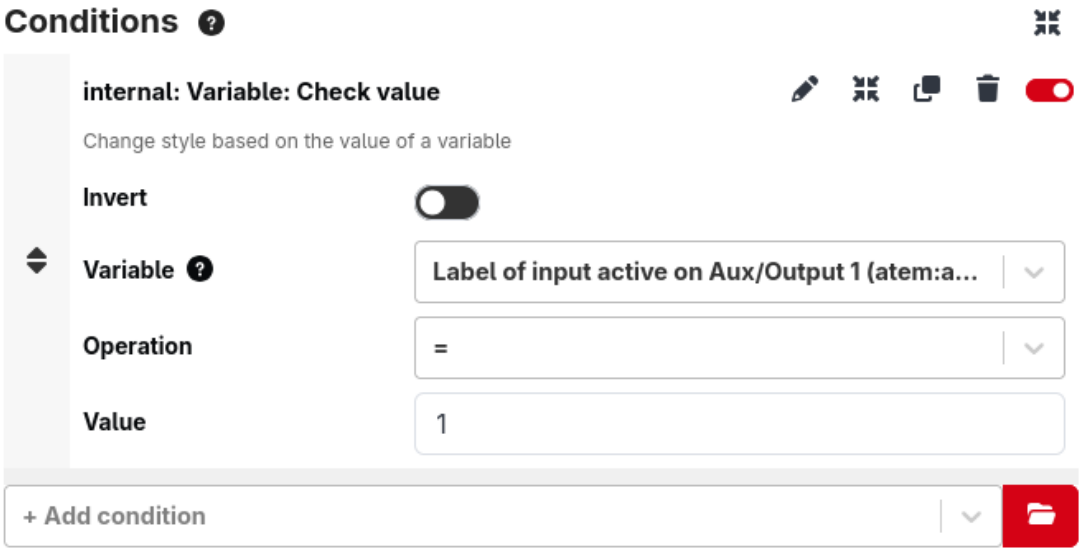


그림 2-50. 조건

이벤트로 트리거가 동작하면 그 시점에 내장(internal) 조건을 다시 검사합니다. 예를 들어 **변수: 불리언 표현식 확인(Variable: Check boolean expression)** 조건에 randomInt(1, 10) > 7을 사용하면 이벤트가 발생할 때마다 난수를 다시 뽑습니다. 이를 이용해 트리거가 일부 경우에만 실행되도록 할 수 있습니다. 연결의 피드백을 사용하는 조건은 해당 연결이 마지막으로 보고한 값을 사용합니다.

액션

트리거가 실행되고 조건을 만족했을 때 수행할 작업을 정의합니다.
버튼에서 사용할 수 있는 모든 액션을 여기에서도 사용할 수 있습니다. **버튼: 누르기 및 놓기 실행(Button: Trigger press and release)** 액션으로 버튼을 누르게 할 수도 있습니다. **누름 유지 시간(ms)(Hold duration (ms))** 옵션을 사용하면 일정 시간 누른 뒤 놓을 수 있어 길게 누르기 액션이 있는 버튼에 유용합니다. 즉시 놓으려면 0으로 둡니다.



그림 2-51. 액션

2.18 변수

변수를 사용하면 액션 옵션, 피드백 옵션, 버튼 스타일 등에 동적으로 변하는 값을 넣을 수 있습니다.

대부분의 변수는 변수(Variables) 페이지에서 유형별로 확인하고 값을 변경할 수 있습니다.

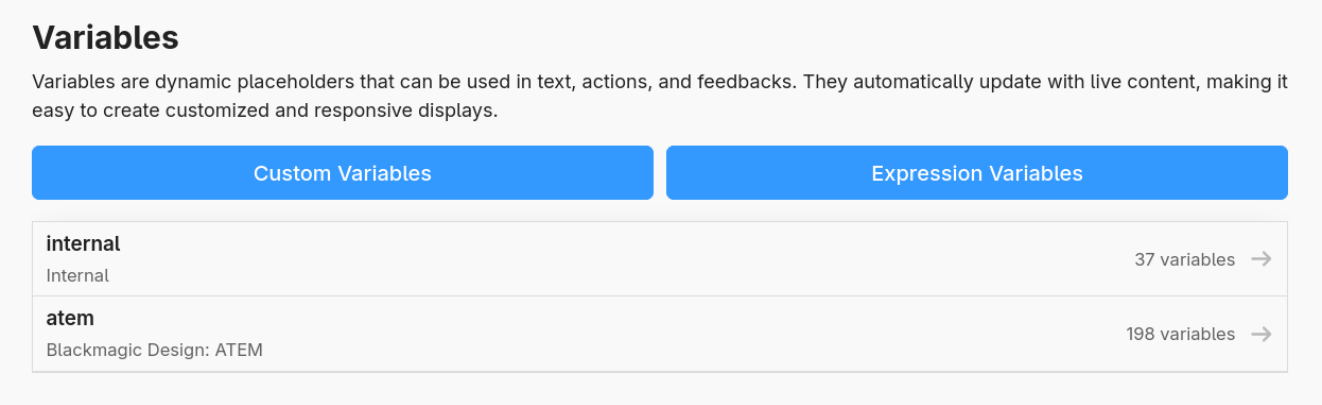


그림 2-52. 변수 페이지

도움말

변수 지원 아이콘이 있는 입력란에서 변수를 사용할 수 있습니다. 다른 입력란에서는 변수를 지원하지 않을 가능성이 높습니다.



그림 2-53. 변수 지원 아이콘

버튼 텍스트에 변수를 사용한 예시는 다음과 같습니다.

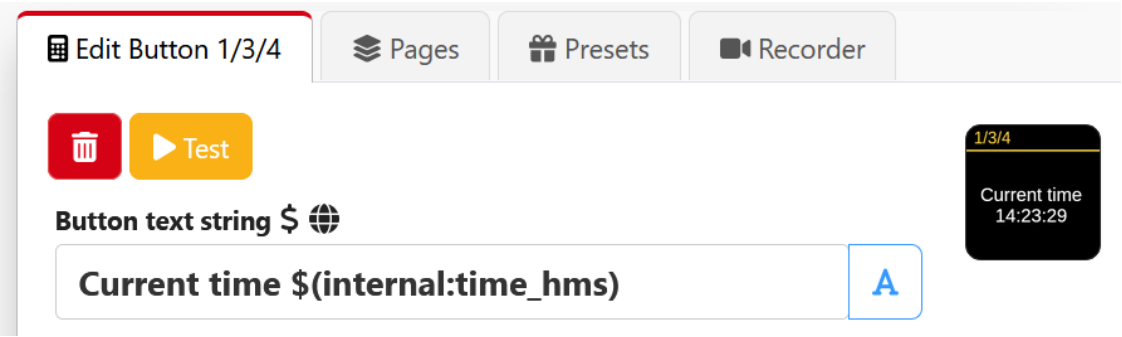


그림 2-54. 연결 변수 사용 예

변수에는 다음과 같은 유형이 있습니다.

연결 변수

설정한 연결에서 제공하는 읽기 전용 변수입니다. 모든 연결이 변수를 제공하는 것은 아니며, 제공하는 변수는 이곳에 표시됩니다.

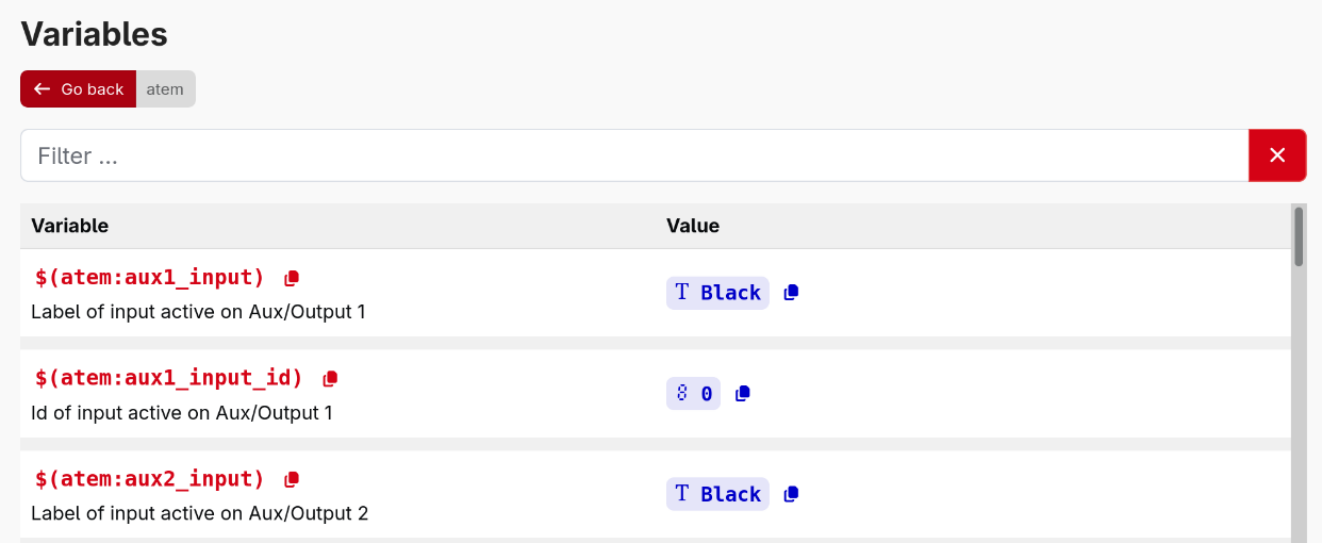


그림 2-55. 연결 변수

참고

이곳에는 'internal'이라는 변수 구역도 있습니다. 연결 변수와 비슷하지만 Companion 자체에서 제공합니다. 현재 날짜와 시간(예: $(internal:time_hms), 월 이름을 나타내는 $(internal:date_monthname)), 연결 및 제어 장치 상태 등이 포함됩니다.

사용자 지정 변수

사용자가 직접 정의하는 변수입니다. 원하는 값을 넣을 수 있으며, 값을 변경하는 내부 액션도 제공됩니다.

다양한 용도로 사용할 수 있습니다.

- 복잡한 동작을 구성하면서 상태 추적
- 변경 가능성이 있는 값을 별칭으로 관리하여, 값이 바뀔 때 수백 곳의 사용 위치를 일일이 수정하는 작업 절약
- 그 밖의 여러 용도

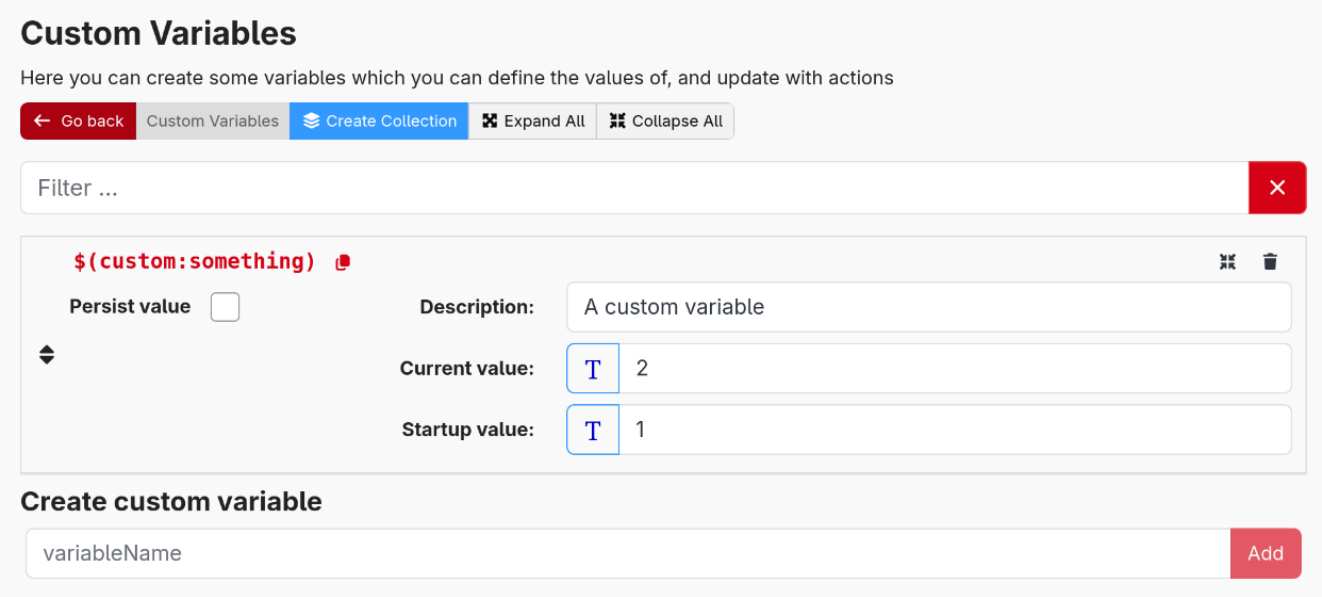


그림 2-56. 사용자 지정 변수

각 사용자 지정 변수에서 다음 항목을 확인하고 설정할 수 있습니다.

- **현재 값(Current value)** 변수의 현재 값입니다.
- **시작 값(Startup value)** Companion을 다시 시작할 때 사용할 값입니다.
- **값 유지(Persist value)** 현재 값을 저장하여 다음 시작 때 사용할지 설정합니다. 디스크 입출력이 늘어납니다.

연결 변수와 마찬가지로 사용자 지정 변수에도 서로 다른 데이터 형식, 즉 자료형의 값을 저장할 수 있습니다.
사용할 수 있는 자료형은 다음과 같습니다.

- string: UTF-8로 인코딩된 텍스트입니다. 예: Hello "world" 😀
- number: IEEE754 부동소수점 숫자입니다. 예: 2.4, -8. 내부적으로 64비트를 사용하므로 크기와 정밀도에 한계가 있습니다. 정수로 사용할 때의 범위는 -2^53 + 1부터 2^53 - 1까지입니다.
- boolean: 가장 작은 디지털 정보인 true 또는 false입니다.
- object: 객체나 다른 자료형을 모아 놓은 값입니다. 기본적으로 키를 사용하는 모음과 인덱스를 사용하는 모음이 있습니다. 전자는 흔히 JSON(JavaScript object notation)이라고 하고, 후자는 배열(Array)이라고 합니다.
  JSON은 중괄호 안에 키 이름과 값으로 이루어진 속성을 쉼표로 구분하여 넣습니다. 예: {"key1":"value1", "key2":42, "third_key":{"description":"objects can be nested"}}
  배열은 대괄호 안에 키 없이 값들을 쉼표로 구분하여 넣습니다. 값은 위치로 참조하며 첫 번째 위치의 인덱스는 0입니다.
  예: ["value1", 42, {"description":"arrays can hold different data types"}]

- null: 문자열이나 숫자 등 다른 자료형으로 분류할 수 없는 특별한 값으로, 별도의 자료형을 갖습니다.
  값과 자료형 모두 null이라고 하며, 값이 유효하지 않음을 나타낼 때 자주 사용합니다.

현재 값이나 시작 값은 Text와 JSON의 두 방식으로 입력할 수 있습니다. 입력란 왼쪽 버튼에 T가 표시되어 있으면 텍스트를 입력합니다. 변수는 해당 텍스트로 갱신되며 자료형은 string이 됩니다.
왼쪽 버튼을 {}로 바꾸면 JSON을 입력할 수 있고 변수는 그에 해당하는 값으로 갱신됩니다. 숫자는 JSON 입력 방식에서 그대로 입력하면 됩니다. 불리언은 true 또는 false를 입력합니다. JSON으로 텍스트를 입력하려면 큰따옴표로 감싸고 텍스트 안의 모든 따옴표를 이스케이프해야 하므로, 텍스트는 Text 방식으로 입력하는 편이 간단합니다.

객체는 하나의 변수에 여러 값을 저장할 때 유용합니다. 예를 들어 3명의 이름을 저장한다고 가정해 보겠습니다.
name1, name2, name3이라는 사용자 지정 변수를 각각 만들어도 되지만, names 변수 하나를 만들고 배열 ["Peter","Paul","Mary"]를 저장할 수도 있습니다. 버튼에 이름을 표시할 때는 표현식으로 해당 값에 접근해야 합니다.
두 번째 이름(인덱스 1)을 가져오는 표현식은 $(custom:names)[1]입니다. 액션으로 배열 안의 이름 하나를 바꿀 때도 표현식을 사용합니다. 예를 들어 arr=$(custom:names); arr[1] = 'John'; arr 표현식으로 변수를 설정합니다.
이 표현식은 현재 배열을 읽고 두 번째 요소를 John으로 바꾼 뒤 새 배열을 반환합니다. name2 전용 변수 하나를 바꾸는 것보다 복잡해 보일 수 있지만, 이름이 100개라면 이 방식이 유용합니다.

참고

이전 버전과의 호환성을 위해 모든 사용자 지정 변수는 기존 $(internal:custom_counter) 형식으로도 해석됩니다.
이 사용 방식은 더 이상 권장하지 않으며 향후 Companion 버전에서 제거될 예정입니다.

표현식 변수

사용자가 직접 정의하는 변수입니다. 사용자 지정 변수와 비슷하지만 값을 직접 지정하는 대신 값을 계산할 표현식을 지정합니다.

값이 자동으로 다시 계산되므로 복잡한 수식을 만들거나 변화에 반응하는 동작을 구성할 때 도움이 됩니다.

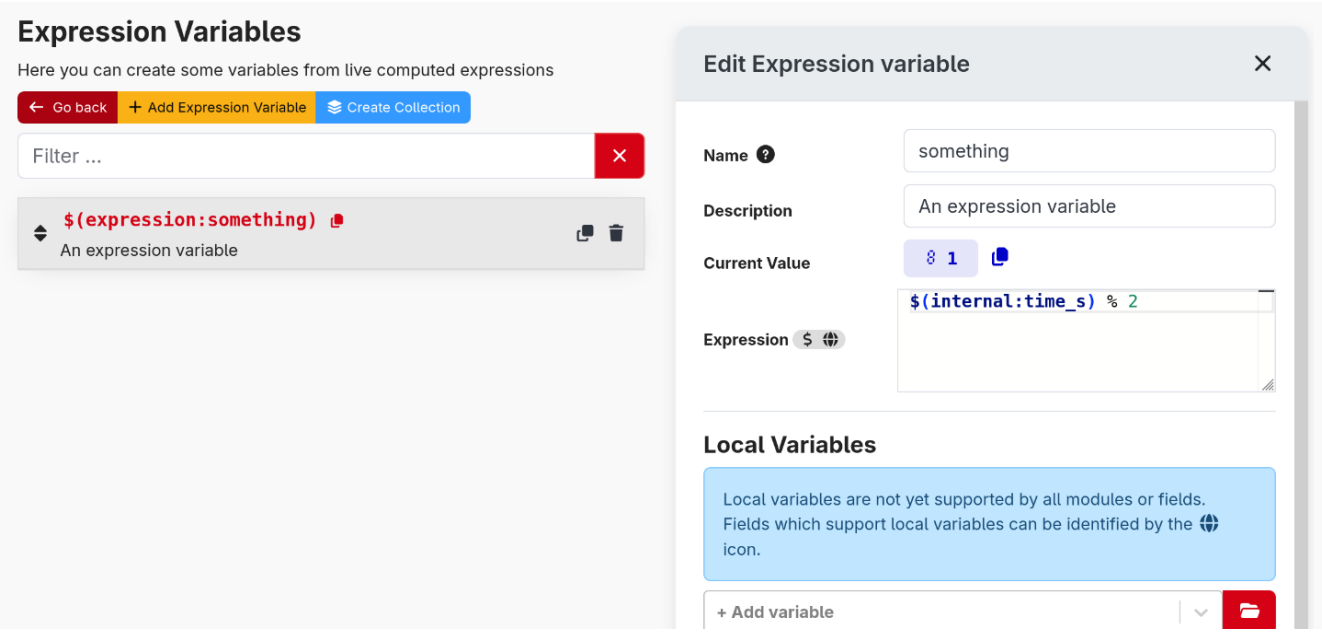


그림 2-57. 표현식 변수

위 그림은 매초 갱신되며 0과 1을 번갈아 표시하는 간단한 표현식 예입니다.

도움말

자세한 도움말은 표현식 문서를 참조하세요.

로컬 변수

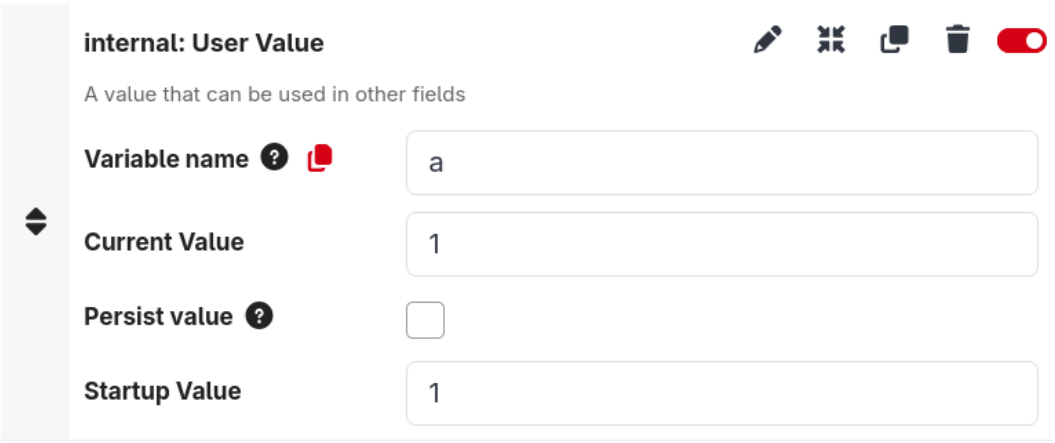
Companion 4.1부터 로컬 변수를 제한적으로 지원합니다. 버튼, 트리거 또는 표현식 변수에 속하는 변수입니다.

값은 해당 변수가 속한 항목 외부에서는 읽을 수 없습니다. 지원되는 입력란에서
$(local: 형식의 이름으로 접근합니다. 사용할 수 있는 곳에는 지구본 아이콘이 표시됩니다.

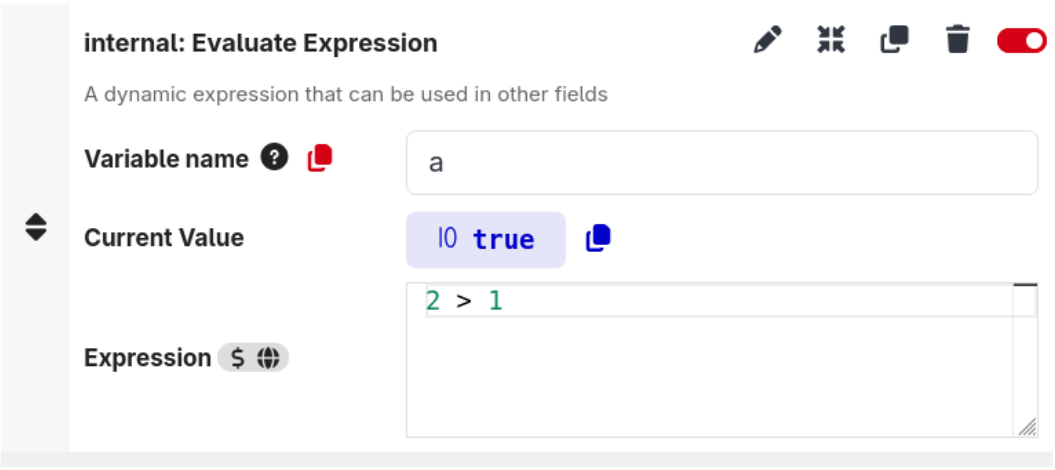
페이지에도 로컬 변수를 정의할 수 있습니다. 해당 페이지의 모든 버튼이 공유하며
$(page: 형식의 이름으로 사용합니다. 자세한 내용은 페이지 변수를 참조하세요.

로컬 변수는 여러 방식으로 값을 얻을 수 있습니다.

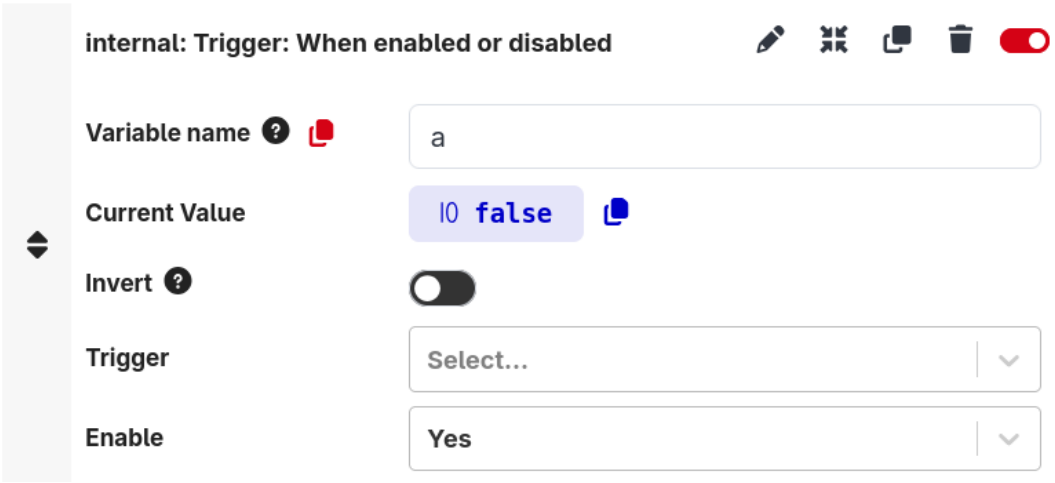
사용자 값(User Value)
사용자 지정 변수와 같은 방식으로 동작합니다.



표현식 계산(Evaluate Expression)
표현식 변수와 같은 방식으로 동작합니다.



피드백(Feedback)
연결에서 제공하는 대부분의 피드백을 사용할 수 있어 불리언 논리에 활용할 수 있습니다.



로컬 변수끼리 순환 참조를 만들지 않도록 주의하세요. Companion이 갱신 빈도를 제한하려고 시도하지만 CPU를 불필요하게 많이 사용합니다.

로컬 변수를 접으면 해당 행에 사용 이름(예: $(local:count))과 현재 값이 함께 표시되어 변수 목록의 상태를 한눈에 확인할 수 있습니다.

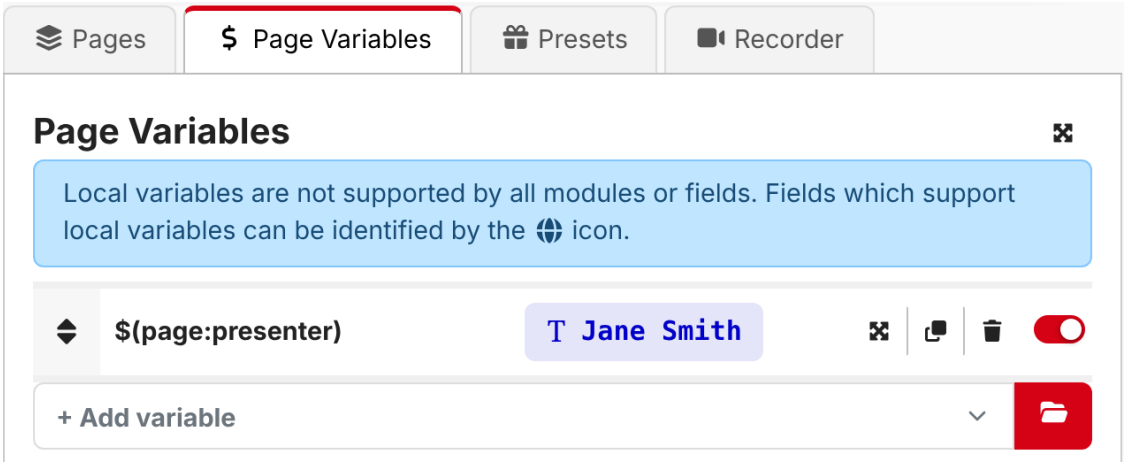


그림 2-58. 현재 값을 보여 주는 접힌 로컬 변수

도움말

이 변수의 값은 소속 항목 외부에서 읽을 수 없지만, 'User Value' 변수의 값은 다른 위치에서 변경할 수 있습니다.

다른 버튼의 로컬 변수 설정

로컬 변수를 설정하는 내부 액션으로 다른 위치의 'User Value' 로컬 변수를 변경할 수 있습니다. 현재 버튼뿐 아니라 **다른 버튼**도 대상으로 지정할 수 있습니다. 버튼 위치(예: 1/0/0 또는 $(this:page)/0/0 같은 표현식)와 변수 이름을 지정하면 됩니다.

값을 설정하는 표현식에서는 **대상** 버튼과 관련된 다음 보조 값을 사용할 수 있습니다.

- $(this:current) - 대상 변수의 현재 값입니다. 값을 전환하거나 증가시킬 때 편리합니다.
- $(target:name) - 대상 위치의 다른 로컬 변수입니다.

예를 들어 1/0/0 위치 버튼의 count 카운터를 증가시키려면 해당 위치와 변수를 대상으로 지정하고 값 표현식을 $(this:current) + 1로 설정합니다.

일부 위치에서는 $(this: 이름 아래에 다음 내장 로컬 변수도 제공됩니다.

변수	값
this:page	버튼의 페이지 번호
this:page_name	현재 페이지 이름
this:column	버튼의 열 번호
this:row	버튼의 행 번호
this:location	버튼 위치(예: 1/2/3)
this:active	현재 버튼이 눌려 있는지 여부
this:step	버튼의 현재 단계
this:step_count	버튼의 단계 수
this:button_status	버튼 상태
this:actions_running	버튼에서 액션이 실행 중인지 여부
this:surface_id	액션을 실행시킨 제어 장치의 ID. 액션에서만 사용 가능
this:delta	회전 액션의 회전량(양수 = 오른쪽/시계 방향, 음수 = 왼쪽/반시계 방향). 회전 액션에서만 사용 가능
this:current	변수를 설정할 때의 현재 값
this:value	피드백의 스타일 재정의에서 사용하는 해당 피드백의 현재 값

2.19 이미지 라이브러리

이미지 라이브러리(Image Library)는 버튼에 사용할 이미지를 한곳에 저장하는 기능입니다. 이미지를 한 번 업로드하고 이름을 지정하면 원하는 만큼 여러 버튼에 사용할 수 있습니다. 이미지를 바꿔야 할 때도 한곳에서 수정하면 사용하는 모든 버튼에 반영됩니다.

사이드바의 **Image Library**에서 열 수 있습니다.

이미지 추가

라이브러리에 이미지를 넣는 방법은 두 가지입니다.

- 이미지 파일 하나 또는 여러 개를 Image Library 페이지에 직접 **드래그 앤 드롭**합니다.
- **업로드(upload)** 버튼을 눌러 컴퓨터에서 파일을 선택합니다.

PNG, JPEG, SVG 등 일반적인 이미지 형식을 사용할 수 있습니다. Companion은 버튼에 맞춰 이미지를 확대·축소하므로 버튼 배경에는 보통 72×72px 이상의 정사각형 이미지가 가장 잘 맞습니다.

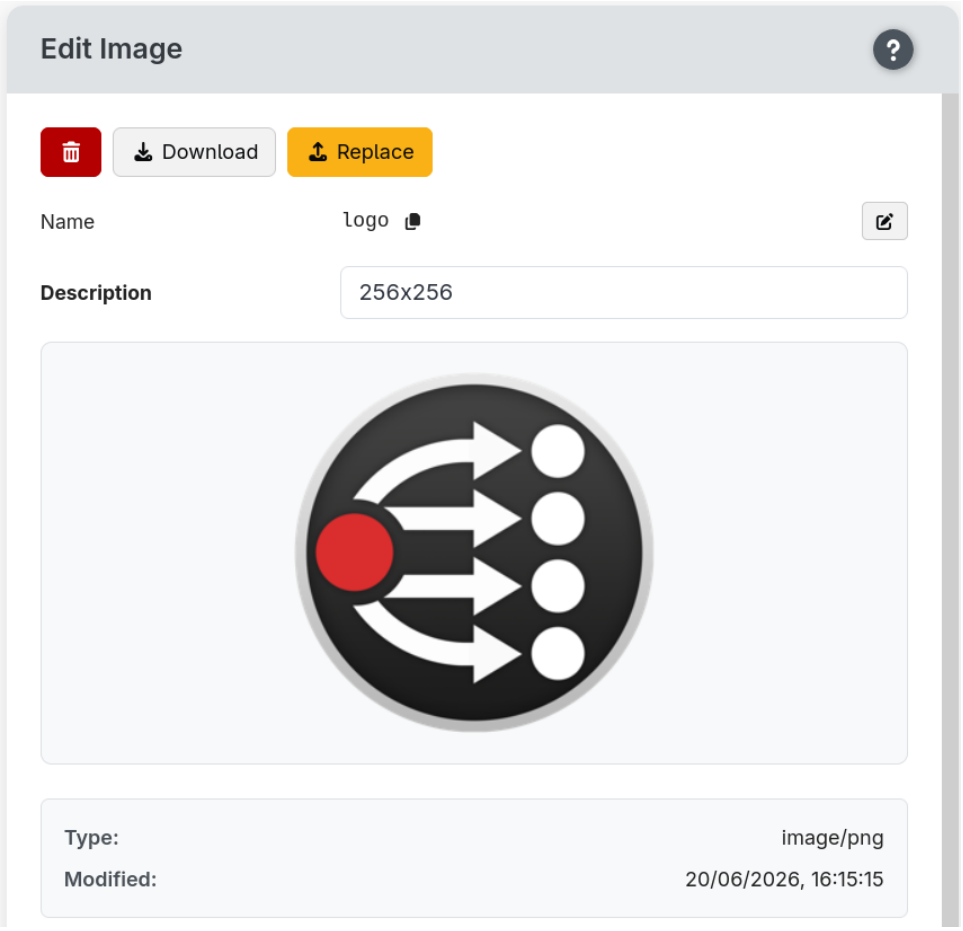


그림 2-59. 이미지 라이브러리 편집

이미지 관리

라이브러리의 각 이미지에서 다음 작업을 할 수 있습니다.

- **이름 변경(Rename)** - 나중에 이미지를 참조할 때 이 이름을 사용합니다. camera-1, logo처럼 기억하기 쉬운 이름을 지정합니다.
- **설명 추가(Add a description)** - 용도를 기억할 수 있도록 선택적으로 메모를 남깁니다.
- **교체(Replace)** - 이름은 유지하면서 새 이미지 파일로 바꿉니다. 사용하는 모든 버튼에 자동으로 반영됩니다.
- **다운로드(Download)** - 이미지를 컴퓨터에 다시 내려받습니다.
- **삭제(Delete)** - 이미지를 삭제합니다.

이미지를 **컬렉션(collections)**으로 묶어 많은 이미지를 깔끔하게 정리할 수도 있습니다.

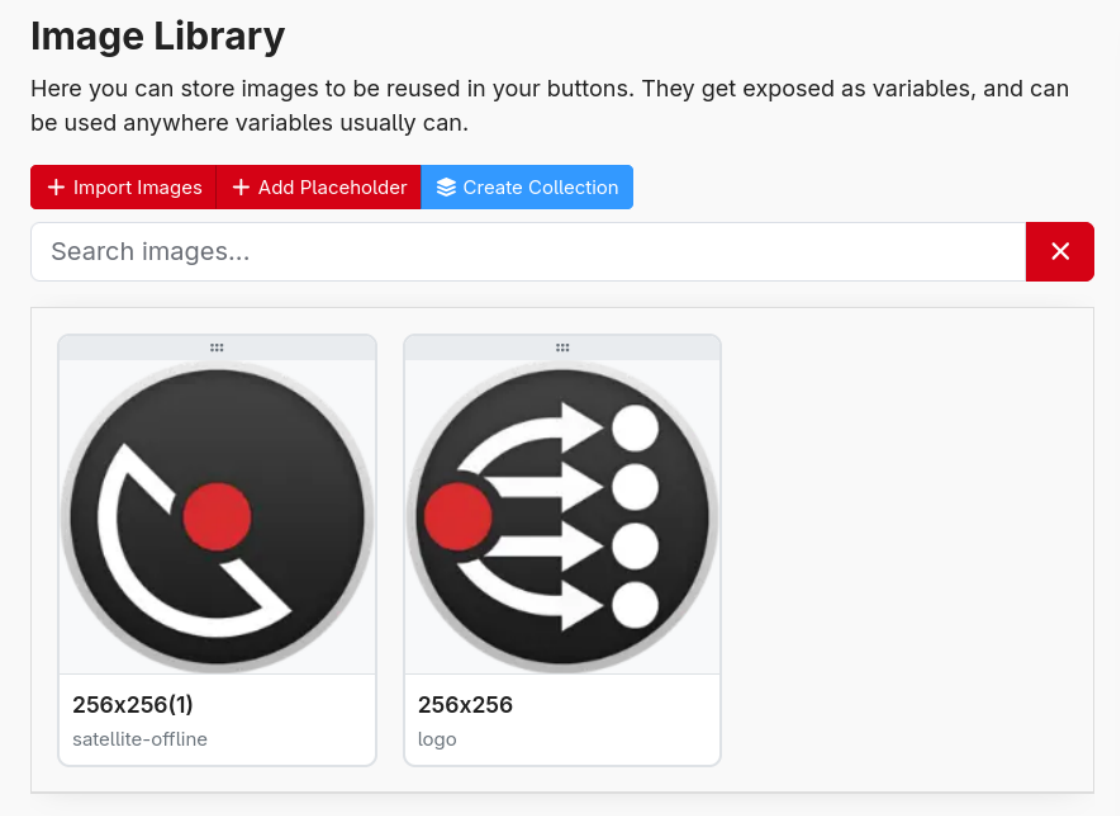


그림 2-60. 이미지 라이브러리 관리

버튼에서 라이브러리 이미지 사용

버튼에 이미지 요소를 추가할 때 이미지 선택기에서 파일을 개별 업로드하는 대신 라이브러리의 이미지를 선택할 수 있습니다.

내부적으로 라이브러리 이미지는 $(image:name) 구문으로 이름을 참조합니다. 이름이 logo인 이미지는 $(image:logo)로 참조합니다. 보통 선택기가 자동 입력하므로 직접 적을 필요는 없습니다. 이 구문 덕분에 표현식을 지원하는 곳에서도 라이브러리 이미지를 사용할 수 있습니다.

도움말

같은 파일을 여러 버튼에 각각 업로드하는 대신 라이브러리에서 참조하면 일관된 모양을 가장 쉽게 유지할 수 있습니다. 라이브러리 이미지를 한 번 수정하면 전체 배치에 반영됩니다.

2.20 모듈

Companion의 연결(Connection)과 제어 장치(Surface)는 모두 모듈을 기반으로 작동합니다.
연결 모듈은 외부 장치나 소프트웨어를 제어하고, 제어 장치 모듈은 물리적 또는 가상 입력 장치를 지원합니다. 예전에는 모듈이 Companion에 내장되어 있었지만, 연결 모듈은 4.0부터, 제어 장치 모듈은 4.3부터 다운로드 가능한 플러그인으로 제공됩니다. Companion 본체와 별개로 업데이트할 수 있습니다.

모듈은 계속 추가되고 업데이트됩니다. 지원 장치와 모듈의 전체 목록은 지원 목록에서 확인할 수 있습니다.

대부분의 모듈은 커뮤니티에서 만들고 관리합니다. 사용하는 장치나 소프트웨어가 목록에 없다면
다음 방법을 고려할 수 있습니다.

1. 코딩 경험이 있다면 직접 모듈을 만들 수 있습니다. 모든 기여를 환영합니다. 자세한 내용은 모듈 개발 안내를 참조하세요.
2. 커뮤니티 포럼의 일부 구성원에게 유료로 모듈 개발을 의뢰할 수 있습니다.
3. Bitfocus 팀은 소규모이지만 일부 모듈 개발을 맡을 수 있습니다. 문의하여 가격과 납기를 확인하세요.
4. 오픈 소스 커뮤니티에 요청을 등록하고 개발을 맡아 줄 사람을 기다릴 수도 있습니다.

모듈 목록

모듈 페이지 왼쪽 패널에는 현재 Companion에 설치된 모듈 목록이 표시됩니다.
표에서 필터와 검색을 사용하여 표시할 모듈을 정할 수 있습니다. 모듈을 선택하면 오른쪽 패널에 상세 정보가 열립니다.

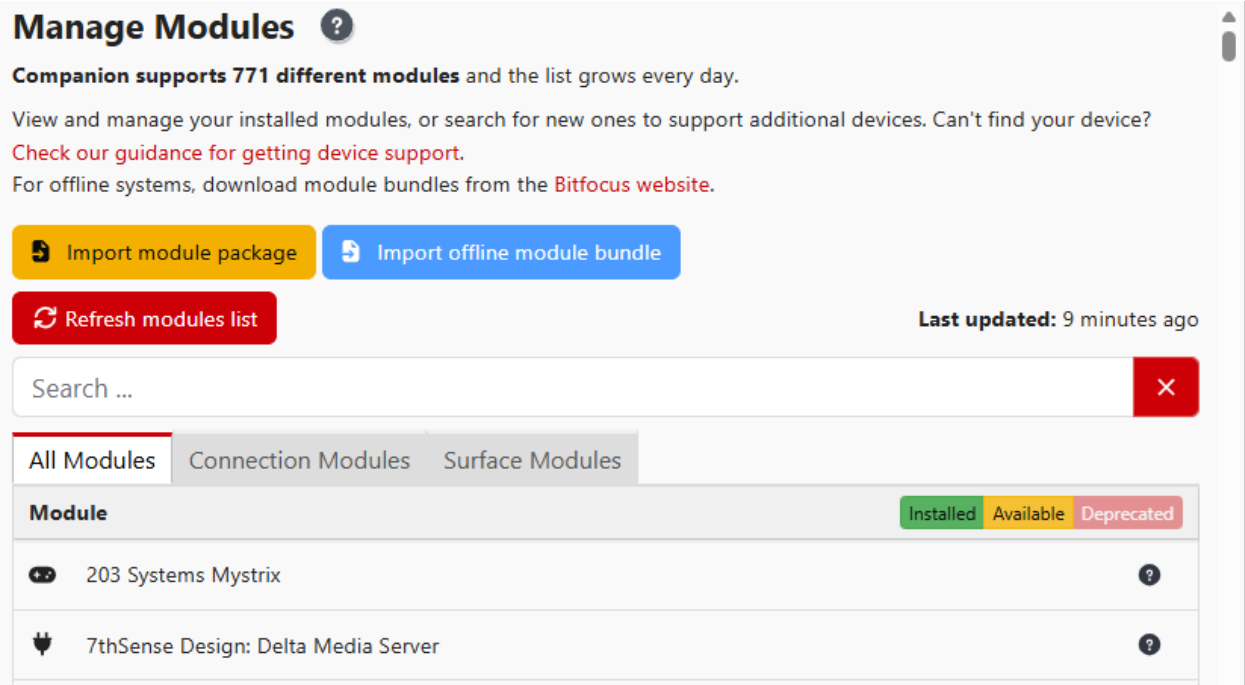


그림 2-61. 모듈 버전

'모듈 패키지 가져오기(import module package)' 버튼으로 모듈 패키지를 가져올 수 있습니다.
파일은 단일 모듈을 빌드하여 생성한 .tgz 형식이어야 하며, 일반적으로 모듈 제작자가 배포합니다.
이 기능으로 모듈의 시험 빌드나 회사·조직 내부 전용 모듈을 쉽게 가져올 수 있습니다.

'오프라인 모듈 번들 가져오기(import offline module bundle)' 버튼은 여러 모듈을 한 번에 가져올 때 사용합니다.
Bitfocus 웹사이트에서 오프라인 모듈 번들을 다운로드할 수 있습니다. 스토어에 직접 접속할 수 없는 오프라인 설치

환경을 위한 기능입니다. 번들은 Companion과 같은 버전 체계를 사용하므로 포함된 모듈이 특정 Companion 버전에서 작동하는지 확인할 수 있습니다.

모듈 관리

이 패널에서는 설치된 모듈 버전을 관리합니다.

기본적으로 표에는 확인된 모든 정식 버전이 표시되며, 각 버전의 설치 여부를 확인할 수 있습니다.
설치된 버전 옆의 플러그 아이콘은 해당 버전을 사용하는 연결이 있다는 뜻입니다.
오른쪽 위에서는 베타 버전과 비권장 버전의 표시 여부를 전환할 수 있습니다.

관리자가 모듈을 비권장으로 지정하면 목록에 **Deprecated** 표시가 나타나고 이 패널의 경고에 이유가 설명됩니다.
자세한 내용은 사용이 권장되지 않는 모듈을 참조하세요.

설치하려는 새 버전이 있다면 '새로고침(refresh)' 버튼으로 스토어의 모듈 정보를 다시 불러올 수 있습니다.
새 버전이 있음을 알릴 수 있도록 이 작업은 일정 간격으로 자동 수행되기도 합니다.

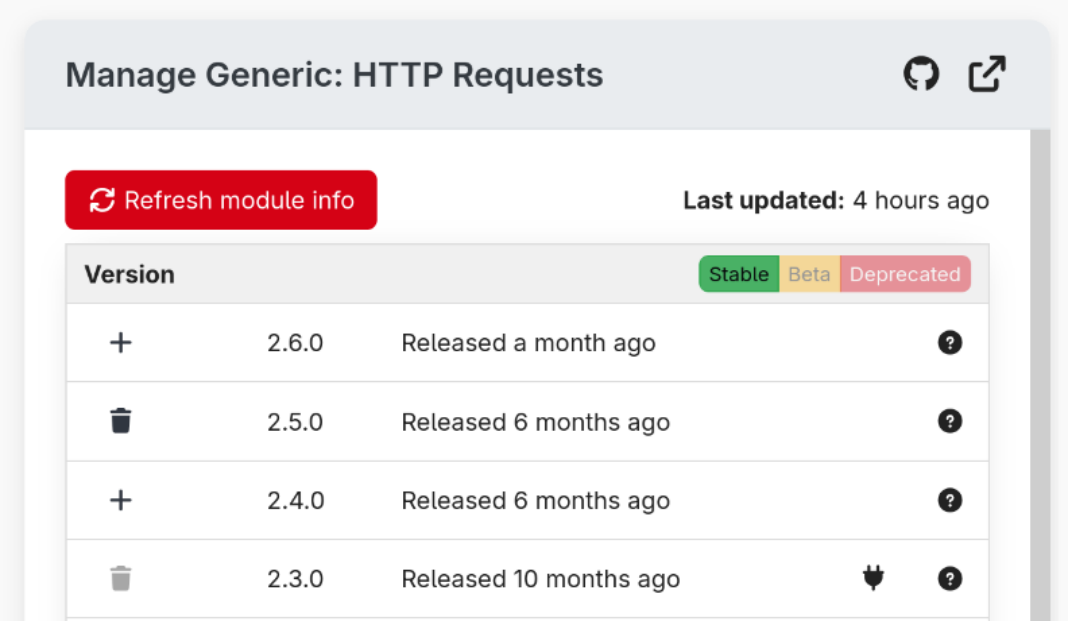


그림 2-62. 모듈 버전

2.21 설정

설정(Settings) 탭에서 Companion의 동작을 설정합니다. 변경 사항은 즉시 적용되므로 저장 버튼이 없습니다.
대부분의 항목 옆에는 기본값으로 되돌리는 초기화 버튼이 있습니다.

설정 메뉴의 **설정 마법사(Configuration Wizard)**를 사용하면 처음 설치했을 때처럼 주요 설정을 다시 안내받을 수 있습니다.

일반

설치 설정

- **설치 이름(Installation Name)**
  이 Companion 설치의 이름입니다. 웹 화면 상단과 브라우저 제목에 표시되며, 네트워크와 제어 장치에 Companion을 알릴 때 사용합니다. 여러 Companion 제어 장치가 있는 네트워크에서 브라우저 탭을 구분할 때 유용합니다.
  COMPANION_INSTALL_NAME 환경 변수로 이름을 지정했다면 여기에서는 변경할 수 없습니다. 서버 구성을 참조하세요.

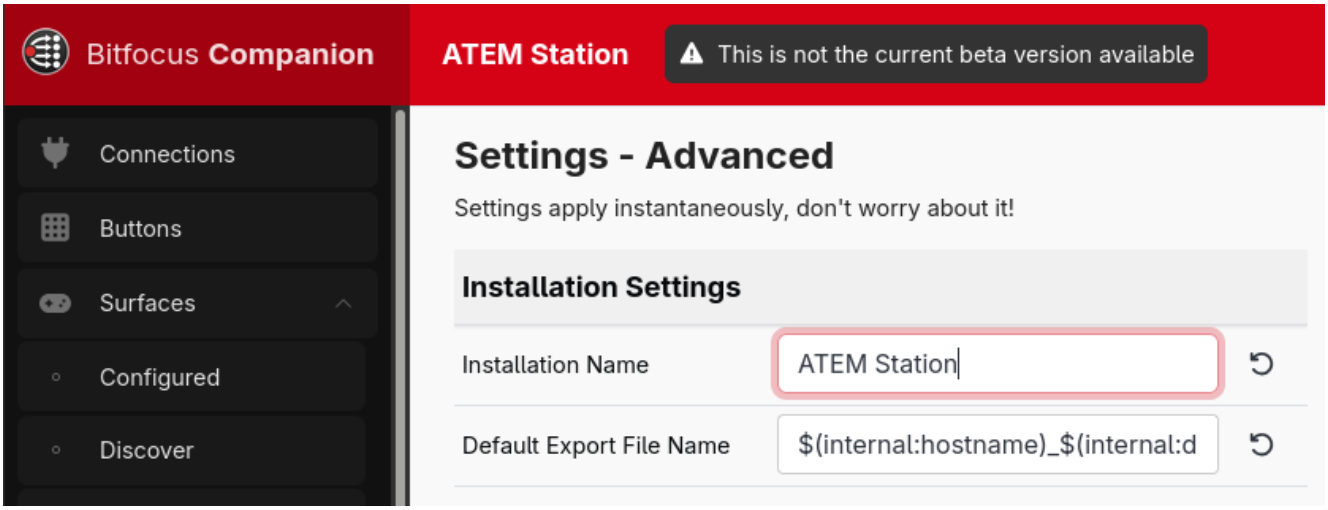


그림 2-63. 설치 이름

- **네트워크에 Companion 알림(Announce Companion on the network (mDNS/Bonjour))**
  Companion Satellite 같은 다른 소프트웨어와 장치가 이 Companion을 자동으로 찾을 수 있게 합니다.
- **기본 내보내기 파일 이름(Default Export File Name)**
  내보낼 때 제안할 파일 이름입니다. 날짜 등의 변수를 포함할 수 있습니다.
- **기본 내보내기 형식(Default Export Format)**
  전체 설정, 페이지, 트리거를 내보낼 때 기본으로 선택할 파일 형식입니다. **압축(Compressed (Default))**, **JSON(JSON (Standard))**, **YAML(YAML (More human readable))** 중에서 선택합니다.
  내보낼 때마다 다른 형식을 선택할 수도 있습니다. 가져오기 / 내보내기를 참조하세요.
- **시간 변수 및 트리거의 시간대(Timezone for time variables and triggers)**
  시간·날짜 내부 변수, 날짜 관련 표현식 함수, 시간 기반 트리거 이벤트에 사용할 시간대입니다.
  기본적으로 Companion을 실행하는 컴퓨터의 시간대를 따릅니다.

데이터 수집

- **상세 사용 통계 사용(Enable detailed usage statistics)**
  Companion 팀이 개발 우선순위를 정할 수 있도록 사용 방식에 관한 익명 정보를 공유합니다.
  수집되는 데이터 보기(View data being collected)를 클릭하면 정확히 어떤 정보를 공유하는지 확인할 수 있습니다.

버튼

- **페이지 위/아래 버튼의 증감 방향 반전(Flip counting direction on page up/down buttons)**
  선택하지 않으면 **Page Up** 버튼을 누를 때 다음 페이지로 증가합니다(2페이지 → 3페이지). 선택하면 이전 페이지로 감소합니다(2페이지 → 1페이지).
- **페이지 위/아래 버튼에 화살표 대신 +와 - 표시(Show + and - instead of arrows on page up/down buttons)**
  페이지 버튼의 기본 화살표 기호를 +와 - 기호로 바꿉니다.
- **버튼의 기본 장식(Default decoration to show on each button. This can be overridden per-button)**
  버튼 주위에 표시할 요소를 정합니다. **상단 표시줄(Top bar)**은 버튼 위쪽에 위치가 적힌 노란색 표시줄을 표시합니다.
  누를 때 테두리(Border (when pressed))는 대신 버튼을 누르는 동안 테두리를 표시합니다.
  없음(None)은 둘 다 표시하지 않습니다. 버튼마다 별도로 지정할 수 있으며, 캔버스 및 버튼 표시를 참조하세요.
- **버튼 상태 아이콘 표시(Show the status icons on each button. This can be overridden per-button)**
  버튼의 연결에 문제가 있거나 액션이 실행 중임을 알리는 작은 아이콘을 표시할지 설정합니다.
  버튼마다 별도로 지정할 수 있습니다.

버튼 격자

- **격자 크기(Grid Size)**
  버튼 격자의 행과 열 수입니다. **격자 크기 구성(Configure Grid Size)**을 클릭하여 변경합니다.
  격자는 4방향 모두로 확장할 수 있으므로 기존 행 위나 열 왼쪽에도 추가할 수 있습니다.
- **새 제어 장치 연결 시 격자 확장 안내(Prompt to expand grid when attaching new surface)**
  격자보다 큰 제어 장치를 연결하면 장치에 맞게 격자를 확장할지 묻습니다.

제어 장치

제어 장치는 이제 제어 장치 통합(Surface Integrations)을 통해 제어합니다. 모든 제어 장치 설정과 통합 옵션은 **Surfaces** 페이지로 이동했습니다. 설정 메뉴에서 **Surfaces**를 클릭하면 해당 페이지의 **제어 장치 통합 및 일반 설정(Surface Integrations and General Settings)** 패널로 이동합니다. 지원 장치에 대한 자세한 내용은 제어 장치 장을 참조하세요.

- **새 USB 장치 감시(Watch for new USB Devices)**
  활성화하면 새로 연결되는 USB 제어 장치를 감시합니다. 비활성화하면 새 Stream Deck을 사용하기 위해 직접 새로고침해야 합니다.
- **새로 검색된 제어 장치 자동 활성화(Auto-enable newly discovered surfaces)**
  제어 장치를 찾는 즉시 Companion에서 사용할지 설정합니다. 일부 제어 장치를 다른 소프트웨어에서도 사용한다면 이 옵션을 끄고 Companion이 사용할 장치를 직접 선택하세요.

네트워크 제어 장치 검색과 Elgato Stream Deck 소프트웨어 플러그인은 이제 이곳의 설정 대신 제어 장치 통합에서 관리합니다.

PIN 잠금

이 설정도 **Surfaces** 페이지에 있습니다.

- **PIN 잠금 사용(Enable PIN Lockout)**
  일정 시간이 지나면 제어 장치를 잠그고 PIN을 입력해야 해제되도록 합니다.
- **잠금 연동(Link Lockouts)**
  한 제어 장치가 잠기면 모든 제어 장치를 잠급니다.
- **잠금 해제 PIN(PIN Unlock Code)**
  제어 장치의 잠금을 해제할 때 입력할 PIN입니다.
- **PIN 잠금 대기 시간(PIN Lockout Timeout (seconds, 0 to turn off))**
  제어 장치에서 아무 조작 없이 이 시간이 지나면 잠깁니다. 단위는 초입니다. 사용하지 않는 동안 자동으로 잠기지 않게 하려면 0을 입력합니다. 대신 필요할 때 잠그는 액션을 버튼에 추가할 수 있습니다.

프로토콜

Companion을 제어하는 데 사용할 수 있는 프로토콜입니다. 페이지 오른쪽 패널에 사용 방법이 설명되어 있으며,
자세한 내용은 이 설명서의 원격 제어를 참조하세요.

일부 기존 프로토콜에는 기본 스위치 아래에 **Deprecated ... API** 스위치가 있습니다. 향후 버전에서 제거될 이전 명령을 활성화하는 옵션입니다. 해당 명령이 꼭 필요한 장비나 소프트웨어가 있을 때만 켜세요.

Satellite

- **Satellite TCP 수신 포트(Satellite TCP Listen Port)** 및 **Satellite WebSocket 수신 포트(Satellite Websocket Listen Port)**
  Satellite 클라이언트의 연결을 수신할 포트입니다.
- **버튼 구독 API 사용(Enable Button Subscriptions API)**
  Satellite 클라이언트가 제어 장치의 페이지만 표시하는 대신 Companion의 모든 버튼을 구독할 수 있게 합니다.
  Elgato Stream Deck 플러그인 등 일부 통합 기능을 완전히 사용하려면 필요하지만, 해당 클라이언트가 PIN 잠금과 페이지 체계를 우회할 수 있게 됩니다.

TCP

활성화하면 TCP 메시지를 수신하여 외부 장치에서 Companion을 제어할 수 있습니다.

- **TCP 수신(TCP Listener)**
  선택하면 TCP를 통한 Companion 제어를 허용합니다.
- **TCP 수신 포트(TCP Listen Port)**
  명령을 수신할 포트입니다.

UDP

활성화하면 UDP 메시지를 수신하여 외부 장치에서 Companion을 제어할 수 있습니다.

- **UDP 수신(UDP Listener)**
  선택하면 UDP를 통한 Companion 제어를 허용합니다.
- **UDP 수신 포트(UDP Listen Port)**
  명령을 수신할 포트입니다.

HTTP

- **HTTP API**
  선택하면 HTTP 원격 제어 명령을 통한 Companion 제어를 허용합니다.

REST API

스크립트나 다른 소프트웨어에서 Companion을 구성하는 버전 관리형 API입니다. 기본적으로 비활성화되어 있습니다.

- **REST API**
  선택하면 REST API를 활성화합니다.

활성화하면 **API 키(API keys)** 표가 표시됩니다. REST API에 보내는 모든 요청에는 이 키 중 하나가 포함되어야 합니다.

- **+**를 클릭하여 키를 만듭니다. **이름(Name)**을 지정하고 허용할 작업을 정하는 **권한 범위(Scopes)**를 선택합니다.
  각 범위가 허용하는 작업은 인증을 참조하세요.
- 키는 생성할 때 한 번만 표시됩니다. 다시 볼 수 없으므로 즉시 복사합니다.
- 연필 아이콘으로 이름이나 권한 범위를 변경하고, 휴지통 아이콘으로 키를 폐기합니다. 폐기한 키는 즉시 사용할 수 없게 됩니다.

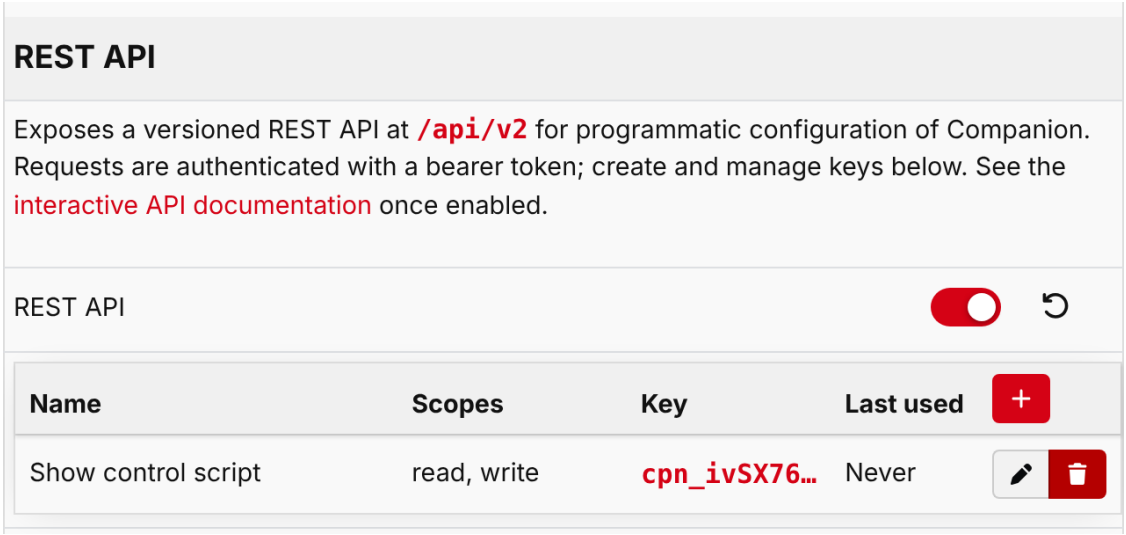


그림 2-64. REST API 설정

Prometheus 지표

기본적으로 비활성화되어 있습니다.

- **Prometheus 지표 엔드포인트(Prometheus metrics endpoint)**
  선택하면 Companion 모니터링을 위한 Prometheus 호환 지표 엔드포인트를 활성화합니다.
  Prometheus로 모니터링을 참조하세요.
- **수집 토큰(Scrape token)**
  Prometheus가 지표를 읽을 때 사용할 토큰입니다. 엔드포인트를 켜면 생성됩니다. 옆의 버튼으로 새 토큰을 만들면 기존 토큰은 사용할 수 없게 됩니다.

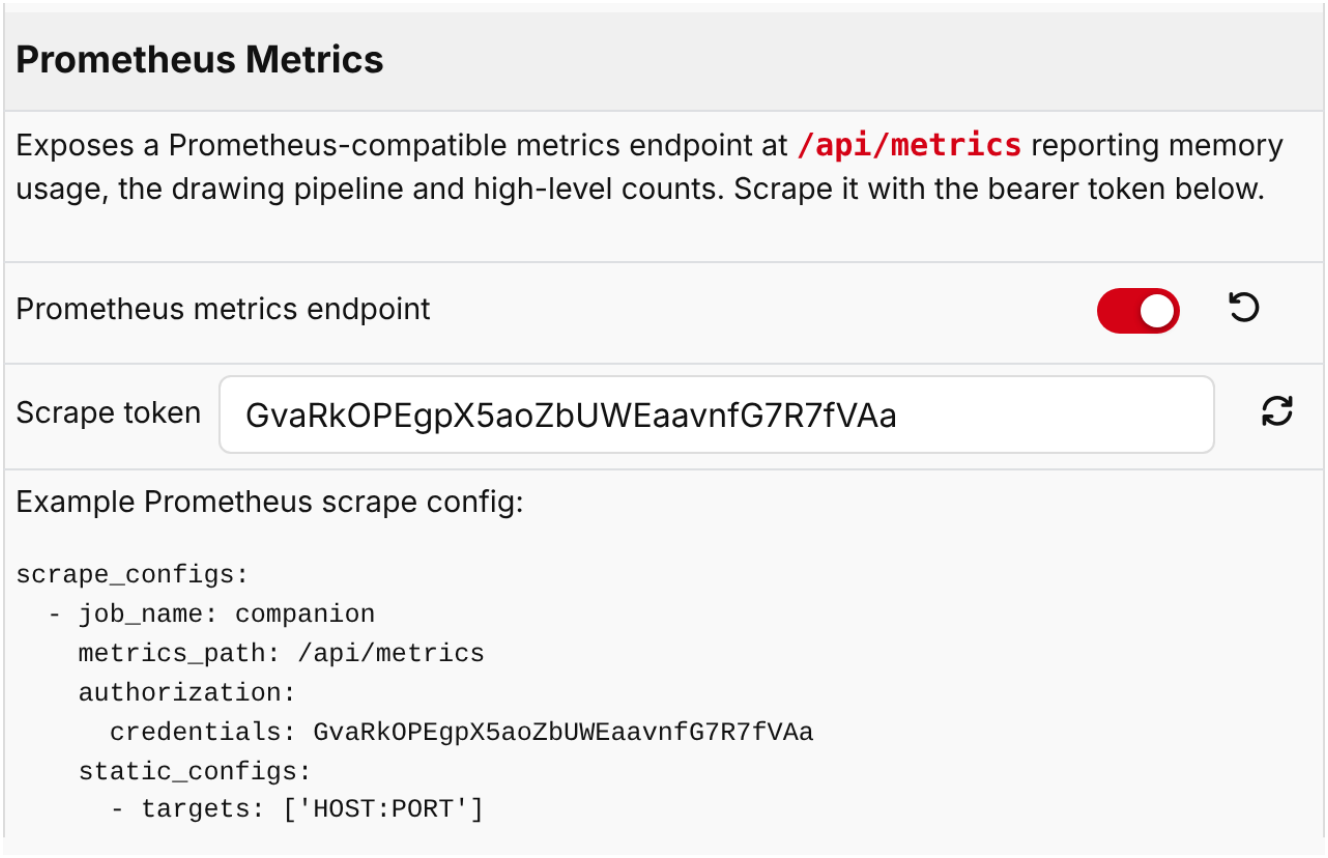


그림 2-65. Prometheus 설정

OSC

활성화하면 OSC 메시지를 수신하여 외부 장치에서 Companion을 제어할 수 있습니다.

- **OSC 수신(OSC Listener)**
  선택하면 OSC를 통한 Companion 제어를 허용합니다.
- **OSC 수신 포트(OSC Listen Port)**
  명령을 수신할 포트입니다.

RossTalk

활성화하면 RossTalk 메시지를 수신하여 외부 장치에서 Companion을 제어할 수 있습니다.

- **RossTalk 수신(RossTalk Listener)**
  선택하면 RossTalk를 통한 Companion 제어를 허용합니다. 포트 7788에서 수신합니다.

Ember+

활성화하면 Ember+ 트리를 제공하여 외부 장치에서 Companion을 제어할 수 있습니다.

- **Ember+ 제공(Ember+ Provider)**
  선택하면 Ember+를 통한 Companion 제어를 허용합니다. 포트 9092에서 수신합니다.

Artnet II 수신

활성화하면 Artnet 메시지를 수신하여 외부 장치에서 Companion을 제어할 수 있습니다. Companion 제어용 GrandMA2 fixture 예제 파일은 해당 탭 아래쪽에서 찾을 수 있습니다.

- **Artnet 수신(Artnet Listener)**
  선택하면 Artnet을 통한 Companion 제어를 허용합니다.
- **Artnet 유니버스(Artnet Universe (first is 0))**
  Companion이 수신할 Artnet 유니버스입니다. 첫 번째 번호는 0입니다.
- **Artnet 채널(Artnet Channel)**
  해당 유니버스에서 Companion이 수신할 시작 채널입니다.

백업

Companion은 일정에 따라 자동 백업할 수 있으며, 원한다면 여러 디렉터리에 저장할 수 있습니다.
이 백업을 클라우드 저장소와 동기화하거나 운영체제 백업에 포함하면 관리자가 더 안심하고 운영할 수 있습니다.

백업 형식은 원시 데이터베이스(Raw Database), 압축(Compressed), JSON, YAML의 네 가지입니다.
원시 데이터베이스 백업이 가장 완전하지만 Companion 화면에서는 복원할 수 없습니다. 압축, JSON, YAML 백업은 Companion 화면에서 복원할 수 있습니다. 아래의 복원 절을 참조하세요.

대부분의 사용자에게는 압축 백업을 권장합니다.

백업 설정

1. **백업 규칙 추가(Add Backup Rule)**를 클릭하여 새 백업 항목을 만듭니다.
2. 오른쪽 패널을 입력합니다.
   - 규칙 이름(Rule Name) - 백업 일정의 이름입니다.
   - Cron 일정(Cron Schedule) - cron 구문으로 백업 일정을 지정합니다. crontab guru에서 올바른 구문을 만드는 데 도움을 받을 수 있습니다.
   - 백업 형식(Backup Type) - 백업 형식을 선택합니다. 대부분의 사용자에게는 압축 백업을 권장합니다.
   - 백업 경로(Backup Path) - Linux·Mac OS에서는 루트 디렉터리, Windows에서는 드라이브 문자를 포함하여 저장 위치를 입력합니다. 비워 두면 Companion의 기본 저장 위치를 사용합니다.
   - 백업 이름 형식(Backup Name Pattern) - Companion 변수를 사용하여 각 백업의 이름을 지정합니다.
   - 유지할 백업 수(Number of Backups to Keep) - 백업 수가 이 값에 도달하면 오래된 백업을 삭제합니다.
3. 노란색 **지금 실행(Run Now)** 버튼을 클릭하여 백업 규칙을 시험합니다.

복원

압축, JSON, YAML 백업은 가져오기/내보내기 페이지에서 복원할 수 있습니다. 설정 가져오기(Import Configuration)를 클릭하고 백업 파일을 선택합니다.

원시 데이터베이스 백업은 Companion 화면에서 가져올 수 없습니다.

백업 규칙을 클릭하면 **백업 규칙 편집(Edit Backup Rule)** 패널 아래쪽에서 이전 백업 목록을 확인할 수 있습니다. Companion 화면에서 백업을 삭제할 수도 있습니다.

고급

관리 화면 비밀번호

활성화하면 설정 페이지를 열 때 비밀번호를 요구합니다. 설치 환경을 안전하게 만드는 기능은 아니며, 가볍게 접속하는 사용자를 막기 위한 기능입니다. 원격 제어 프로토콜이나 REST API는 보호하지 않습니다.

- **잠금 사용(Enable Locking)** 관리자 비밀번호와 잠금 기능을 사용할지 설정합니다.
- **세션 제한 시간(Session Timeout (minutes, 0 for no timeout))** 아무 조작 없이 이 시간이 지나면 화면이 잠깁니다. 단위는 분입니다. 0으로 설정하면 자동으로 잠기지 않습니다.
- **비밀번호(Password)** 화면의 잠금을 해제할 때 입력할 비밀번호입니다.

HTTPS 웹 서버

배포 환경에 필요하다면 Companion 웹 화면용 HTTPS 서버를 활성화할 수 있습니다. Companion 화면을 인터넷에 공개하는 것은 권장하지 않으며, HTTPS를 사용해도 그러한 구성에 추가 보안이 제공되지는 않습니다.

- **HTTPS 웹 서버(HTTPS Web Server)**
  선택하면 HTTPS 웹 서버를 활성화합니다.
- **HTTPS 포트(HTTPS Port)**
  HTTPS를 제공할 포트 번호입니다.
- **인증서 유형(Certificate Type)**
  "자체 서명(Self Signed)"은 내장 인증서 생성기를 사용하고, "외부(External)"는 파일 시스템의 인증서 파일을 연결합니다.

  일반 이름(도메인 이름)(Common Name (Domain Name)) 자체 서명 인증서를 발급할 "Common Name"을 입력합니다. 보통 도메인 이름이나 호스트 이름을 사용합니다.

  인증서 유효 일수(Certificate Expiry Days) 자체 서명 인증서의 유효 기간을 일수로 지정합니다. 기본값은 365일입니다.

  개인 키 파일(전체 경로)(Private Key File (full path)) 외부 개인 키 파일의 전체 경로입니다.

  인증서 파일(전체 경로)(Certificate File (full path)) 외부 인증서 파일의 전체 경로입니다.

  체인 파일(전체 경로)(Chain File (full path)) 외부 인증서 체인 파일의 전체 경로를 선택적으로 지정합니다.

실험 기능

기능을 정확히 이해하지 못한다면 이 설정을 변경하지 마세요.

- **Companion Cloud 탭(Companion Cloud Tab (Deprecated))**
  사용이 권장되지 않는 Companion Cloud 서비스의 **Cloud** 페이지를 메뉴에 표시합니다. 이 설정은 Companion이 아니라 브라우저에 저장됩니다.

2.22 가져오기 / 내보내기

이 탭에서는 설정을 .companionconfig 파일로 내보내거나 가져올 수 있습니다. 설정 백업이나 새 컴퓨터로의 이전에 사용할 수 있으며, 파일에서 한 페이지만 선택하여 가져올 수도 있습니다.

한 페이지만 내보내려면 기본 버튼 페이지의 버튼 격자 위에 있는 **페이지 작업(Page actions)**(...) 메뉴에서 **페이지 내보내기(Export page)**를 선택합니다.

내보낼 때 파일 형식을 선택할 수 있습니다.

- **압축(Compressed (Default))** - 파일 크기가 가장 작습니다. 대부분의 경우 권장합니다.
- **JSON(JSON (Standard))** - 압축하지 않는 형식으로 다른 도구에서도 읽을 수 있습니다.
- **YAML(YAML (More human readable))** - 사람이 읽고 직접 편집하기 편리한 형식입니다.

처음 선택되는 형식은 설정의 **기본 내보내기 형식(Default Export Format)**에서 변경할 수 있습니다.

이곳에서 설정을 완전히 초기화할 수도 있습니다.

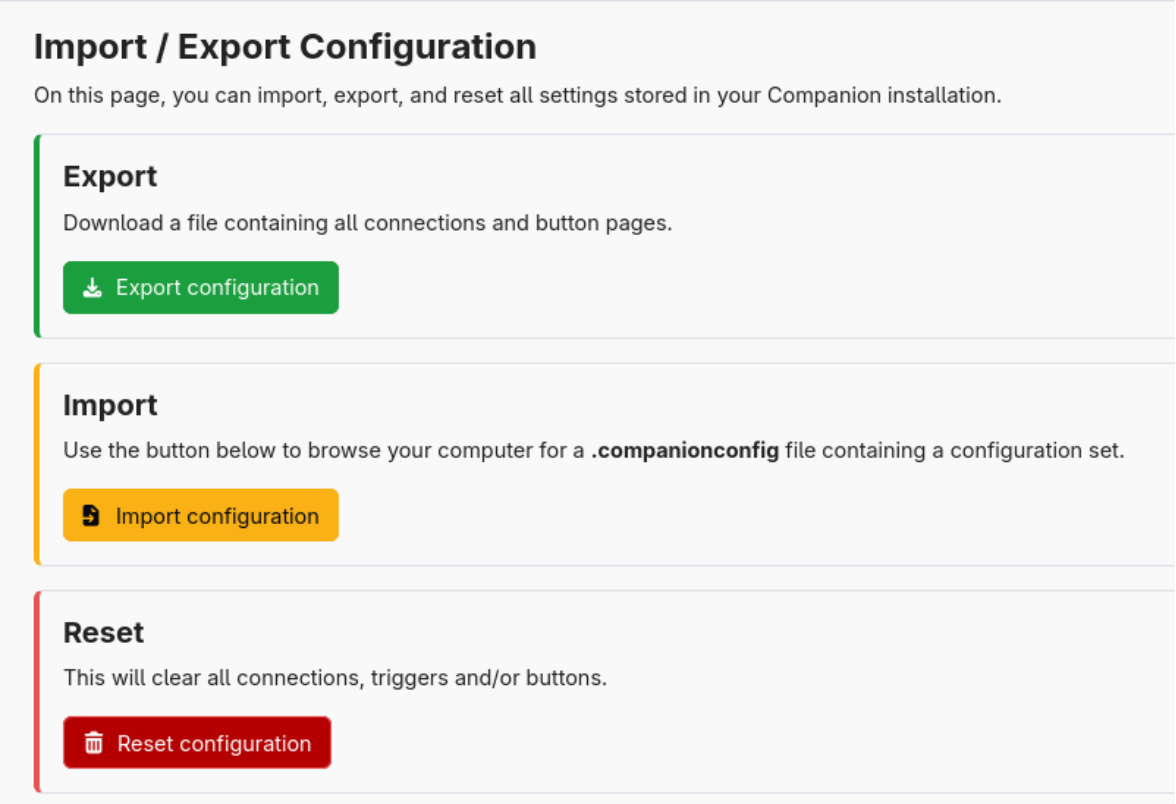


그림 2-66. 가져오기/내보내기

가져오기

.companionconfig 파일을 가져옵니다. 제공한 파일에서 가져올 수 없는 항목의 옵션은 비활성화됩니다.

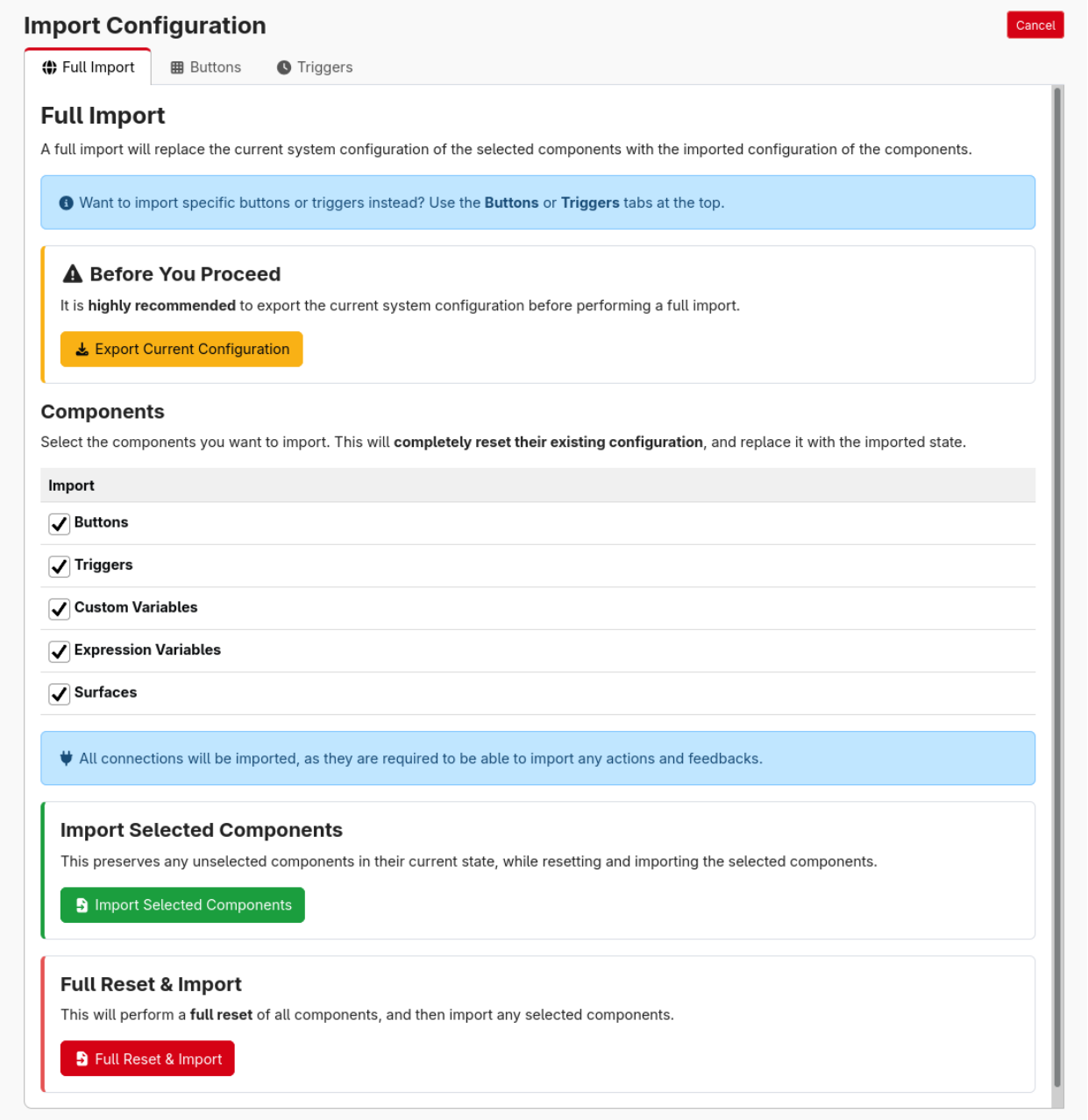


그림 2-67. 가져오기

전체 가져오기(Full import) 탭에서 가져올 범주를 지정합니다. "Import, Preserving unselected components" 버튼은 선택하지 않은 구성 요소를 유지하면서 가져옵니다. 전체 초기화는 하지 않지만 선택한 구성 요소는 초기화한 뒤 파일의 내용으로 채웁니다. "Full Reset then Import" 버튼은 모든 구성 요소를 초기화한 뒤 선택한 설정 범주만 가져옵니다.

실수에 대비하여 먼저 현재 설정을 내보내 백업하세요.

최대 1GB의 설정을 가져올 수 있습니다. 매우 큰 설정에는 YAML 대신 압축 또는 JSON 형식을 사용하세요.
YAML 파일은 전체 내용을 한 번에 읽어야 합니다.

버튼(Buttons) 탭과 **트리거(Triggers)** 탭에서는 일부 페이지나 트리거만 가져올 수 있습니다. 필요하면 연결의 대응 관계도 다시 지정할 수 있습니다.

2.23 로그

로그(Log) 구역에는 Companion의 상태 변화가 표시됩니다. 로그 수준별로 필터링할 수 있습니다.

모듈별 상세 로그는 **연결(Connections)** 페이지에서 확인할 수 있습니다.

세 버튼으로 경고(warnings), 정보(info), 디버그(debug) 메시지의 표시 여부를 전환합니다. 오류는 설정에 관계없이 항상 표시됩니다. 현재 로그 기록을 지울 수도 있으며, 로그나 지원용 번들을 내보내 개발자가 사용자의 설정으로 문제를 진단하도록 도울 수 있습니다.

각 로그 행에는 기록된 날짜와 시간이 밀리초 단위로 표시되므로 발생 순서를 파악하는 데 도움이 됩니다.

문제가 발생하면 이곳을 확인하세요.



그림 2-68. 로그 파일

디버그 메시지를 활성화한 로그 예시

3 표현식

표현식은 Companion에 내장된 작은 수식 언어입니다. 일반 입력란은 텍스트에 변수 값을 넣는 기능만 제공하지만, 표현식은 계산, 값 비교, 조건 판단, 텍스트 서식을 처리합니다. 따라서 별도의 버튼·트리거·외부 스크립트가 필요했던 동작을 표현식으로 구현할 수 있습니다.

다음은 활용 예입니다. 19:30에 시작하는 공연까지 남은 시간을 계산하며 버튼 텍스트로 사용할 수 있습니다.

```
secondsToTimestamp(max(0, timeDiff($(internal:time_hms), '19:30:00')), 'HH:mm:ss')
```

표현식을 사용할 수 있는 곳

Companion의 여러 곳에서 표현식을 사용할 수 있습니다.

- **버튼 텍스트**: 텍스트 요소의 표현식 모드를 켜면 고정 텍스트를 쓰는 대신 계산 결과를 표시합니다.
- **모든 버튼 요소 속성**: 버튼 요소의 색상, 위치, 크기, 표시 여부 등 모든 속성에 표현식을 사용할 수 있습니다.
- **내부 액션**: 예를 들어 **Custom Variable: Set value** 액션의 값을 표현식으로 설정할 수 있습니다.
- **내부 피드백**: **Variable: Check boolean expression**은 표현식이 참일 때 버튼 스타일을 바꿉니다. **Evaluate Expression**은 평가 결과를 스타일 설정에 사용할 수 있게 합니다.
- **표현식 변수와 로컬 변수**: 표현식으로 값이 자동 재계산되는 변수입니다. 변수를 참조하십시오.
- **트리거 조건**: 버튼과 같은 피드백을 사용하므로 불리언 표현식 피드백도 사용할 수 있습니다.
- **모든 액션·피드백 입력란**: Companion 4.3부터 모듈은 **모든** 액션·피드백 입력란에서 표현식을 지원할 수 있습니다. 입력란 옆의 표현식 전환 버튼으로 모드를 바꿉니다.

참고

아직 모든 모듈이 액션·피드백 입력란에서 표현식을 지원하지는 않습니다. 모듈이 새 API로 업데이트되면서 지원 범위가 늘어날 예정입니다.

표현식 모드와 텍스트 안의 변수

가장 혼동하기 쉬운 부분이므로 다음 차이를 알아 두십시오.

- 변수를 지원하는 **일반 텍스트 입력란**에서는 평문을 쓰면 $(...) 자리에 변수 값이 들어갑니다. 예: Cam $(atem:input_1_name).
- **표현식 입력란**에서는 입력 내용 전체가 텍스트가 아닌 하나의 수식입니다. 단어를 그대로 쓸 수 없으며 텍스트는 따옴표로 감싸야 합니다. Cam $(atem:input_1_name)은 구문 오류지만 `Cam ${$(atem:input_1_name)}`는 동작합니다. 아래 템플릿 문자열을 참조하십시오.

두 경우 모두 변수는 $(connection:variable)로 참조하며, 차이는 그 주변 내용을 작성하는 방식에 있습니다.

기본 구성 요소

표현식은 값, 변수, 연산자, 함수 호출, 조건식 등으로 구성합니다. 아래에서 간단히 설명하며, 전체 항목은 연산자와 함수 참조에 있습니다.

값

숫자는 1, 1234, 1234.5678처럼 정수나 부동소수점 수로 쓸 수 있습니다. 16진수나 2진수도 변환하여 처리합니다. 0x10은 16, 0b11은 3입니다.

문자열은 작은따옴표 또는 큰따옴표로 감쌉니다. 예: 'on air', "standby".

불리언값은 true와 false로 씁니다.

변수

사용자 변수와 로컬 변수를 포함한 모든 변수는 일반적인 구문으로 참조할 수 있습니다.

```
$(internal:time_s)
$(custom:my_variable)
```

표현식 안에서 변수는 원래의 자료형을 유지합니다. 숫자 5를 담은 사용자 변수는 숫자로 동작하며, 배열을 담은 변수는 $(custom:my_array)[0]처럼 인덱스로 접근할 수 있습니다. 존재하지 않는 변수의 평가 결과는 undefined입니다.

참고

표현식에서는 $(custom:$(custom:b))처럼 중첩된 변수 참조를 지원하지 않습니다. 필요한 경우 parseVariables 함수를 사용하십시오.

연산자

일반적인 산술·비교·논리 연산자를 사용할 수 있습니다.

```
$(custom:a) + $(custom:b)
($(custom:a) > $(custom:b)) && !$(custom:c)
```

전체 목록은 연산자 참조를 확인하십시오.

주의

여러 언어와 달리 +는 **항상 숫자 덧셈만 수행**하며 문자열을 합치지 않습니다. 'show ' + 'time'의 결과는 NaN입니다. 텍스트를 합치려면 템플릿 문자열이나 concat 함수를 사용하십시오.

함수

문자열 조작, 서식 지정, 시간 처리 등 다양한 함수를 사용할 수 있습니다.

```
round($(custom:a))
secondsToTimestamp($(custom:total_seconds), 'mm:ss')
```

전체 목록은 함수 참조를 확인하십시오.

조건식(삼항)

삼항 연산자 condition ? valueIfTrue : valueIfFalse로 두 값 중 하나를 선택합니다.

```
bool($(obs:recording)) ? 'REC ●' : 'REC'
```

도움말

변수에 실제 불리언값 대신 텍스트 "false" 또는 "0"이 들어 있으면, 비어 있지 않은 문자열이므로 조건에서 _참_으로 처리됩니다. bool 함수로 감싸면 이런 값을 의도에 맞게 처리할 수 있습니다.

여러 경우를 처리하도록 조건식을 이어 쓸 수도 있습니다.

```
$(custom:level) > 90 ? 'HIGH' : $(custom:level) > 50 ? 'MEDIUM' : 'LOW'
```

템플릿 문자열

여러 값을 조합해 텍스트를 만드는 가장 간단한 방법입니다. 백틱으로 감싸며, 각 ${...} 안에 유효한 표현식을 넣을 수 있습니다.

```
`${$(custom:a)} dB`
`Clip ${$(custom:current)} of ${$(custom:total)}`
```

여러 줄과 임시 변수

긴 표현식은 줄바꿈이나 세미콜론(;)으로 구분한 여러 문장으로 나누고, 중간 결과를 임시 변수에 저장할 수 있습니다.
마지막 문장의 값이 표현식의 결과가 됩니다.

```
myval = $(custom:a) + $(custom:b)
myval / 2
```

return으로 반환 결과를 명시하거나 일찍 종료할 수도 있습니다. return myval / 2는 해당 값을 반환하고 표현식을 끝냅니다.
return과 반환할 값은 반드시 같은 줄에 있어야 합니다.

임시 변수는 표현식을 평가하는 동안만 존재합니다. 다른 곳에서는 접근할 수 없으며 사용자 변수와도 별개입니다.

도움말

문장이 값으로 끝나고 다음 줄이 `, (, [로 시작하면 두 줄을 하나의 문장으로 해석할 수 있습니다.
예를 들어 계산 다음 줄에 단독으로 놓인 템플릿 문자열이 윗줄의 일부가 될 수 있습니다. 확실하지 않으면 문장 끝에
세미콜론(;)을 붙여 모호함을 없애십시오.

let과 const로 변수를 명시적으로 선언할 수도 있습니다. 블록과 함수를 사용할 때는 이 차이가 중요합니다.
변수의 생성·갱신·유효 범위는 고급 표현식에서 설명합니다.

주석

블록 주석과 줄 끝 주석을 모두 지원하므로 긴 표현식을 이해하기 쉽게 정리할 수 있습니다.

```
// end of line comment
/* block comment */
```

객체와 배열

표현식으로 객체와 배열을 만들고 사용할 수 있습니다.

```
[1, 2, 3]
{ label: 'Cam 1', input: 1 }
```

속성과 요소는 일반적인 대괄호나 점 표기법으로 접근합니다. 객체·배열을 담은 사용자 변수나 구조화된 데이터를 제공하는 모듈 변수에 특히 유용합니다.

```
$(custom:names)[1]
$(custom:settings)['some-prop']
$(custom:settings).timeout
```

더 깊게 중첩된 구조를 조회하려면 jsonpath 함수를 참조하십시오.

선택적 연결(?.)로 존재하지 않을 수 있는 값을 안전하게 읽고, 펼침 연산자(...)로 배열이나 객체를 합칠 수도 있습니다.

```
$(custom:settings)?.timeout ?? 1000
[...$(custom:list_a), ...$(custom:list_b)]
{ ...$(custom:defaults), label: 'override' }
```

더 활용하기

표현식은 기본 구성 요소 외에도 변수 선언, 제어 흐름(if, 반복문), 사용자 정의 함수, 배열 처리를 이용한 스크립팅을 지원합니다. 수식 하나보다 복잡한 처리가 필요하면 **고급 표현식**을 참조하십시오.

활용 예제

표현식으로 해결할 수 있는 문제의 완전한 예제입니다. 연결 이름인 obs, atem 등은 예시이므로, 자신이 설정한 연결의 변수로 바꾸십시오.

지정 시각까지 카운트다운

버튼 텍스트의 표현식 모드를 켜고 사용하면 19:30까지 남은 시간을 표시하며, 그 이후에는 0을 유지합니다.

```
secondsToTimestamp(max(0, timeDiff($(internal:time_hms), '19:30:00')), 'HH:mm:ss')
```

여러 줄 표현식으로 작성하면 지정 시각이 지난 뒤 텍스트를 완전히 바꿀 수도 있습니다.

```
remaining = timeDiff($(internal:time_hms), '19:30:00')
remaining > 0 ? secondsToTimestamp(remaining, 'mm:ss') : 'ON AIR'
```

방송 중 버튼 깜박이기

Variable: Check boolean expression 피드백으로 배경색을 빨간색으로 바꿉니다.

```
bool($(obs:streaming)) && blink(500)
```

스트리밍 중에는 피드백이 500ms 주기로 켜짐과 꺼짐을 반복하여 버튼이 깜박입니다. 스트리밍 중이 아니면 꺼진 상태를 유지합니다.

방송 중 버튼을 부드럽게 점멸하기

깜박임보다 부드러운 효과를 내려면 버튼의 빨간색 Box 요소에 **불투명도(Opacity)** 값으로 oscillate를 사용합니다.

```
bool($(obs:streaming)) ? oscillate(2000) * 100 : 0
```

스트리밍 중에는 상자가 2초마다 부드럽게 나타났다가 사라집니다. 스트리밍 중이 아니면 숨겨집니다.

여러 값을 순환하기

Custom Variable: Set value 액션의 값 입력란에서 표현식 모드를 켜고 다음을 사용합니다.

```
($(custom:camera) % 4) + 1
```

누를 때마다 변수 값이 1 → 2 → 3 → 4 → 1 순서로 바뀝니다. 버튼 하나로 카메라, 프리셋, 페이지를 차례로 전환할 때 유용합니다.

숫자의 표시 형식 지정

페이더 레벨을 소수점 한 자리와 단위를 포함한 버튼 텍스트로 표시합니다.

```
`${toFixed($(mixer:ch1_level), 1)} dB`
```

클립 번호 앞에 0을 채워 고정 자릿수로 표시합니다.

```
substr(concat('000', $(custom:clip)), -3)
```

진행률을 백분율로 표시

여러 변수를 조합해 읽기 쉬운 상태 정보로 만듭니다.

```
pct = round($(player:position) / $(player:duration) * 100)
`${pct}%`
```

변수가 설정되지 않았을 때 대체 값 사용

?? 연산자는 값이 undefined일 때만 대체 값을 사용합니다.

```
$(custom:operator_name) ?? 'No operator'
```

유효하지 않은 표현식

표현식에 구문 오류가 있으면 유효하지 않은 것으로 처리합니다. 결과를 만들지 않으며 문제를 설명하는 오류 로그를 기록합니다.

변수가 없거나 값이 적합하지 않다는 이유만으로 표현식이 유효하지 않게 되지는 않습니다. 대신 NaN이나 undefined처럼 예측 가능하지만 쓸모없는 결과를 반환하는 경우가 많습니다. 텍스트를 예상했는데 NaN이 나오면 +로 문자열을 합치고 있는지 확인하고, 템플릿 문자열이나 concat을 사용하십시오.

사용한 값이나 함수 때문에 표현식 평가 자체가 실패하는 경우를 발견하면 버그로 보고해 주십시오.

참고

- 표현식 언어는 JavaScript의 일부 기능을 제공합니다. 대부분의 JavaScript 구문은 동작하지만 클래스, 정규식 리터럴, try/catch 등은 사용할 수 없습니다. 함수는 화살표 함수(x => ...)만 지원하며 function 키워드는 지원하지 않습니다.
- 여러 문장은 줄바꿈이나 세미콜론(;)으로 구분해야 합니다.
- 이 기능들을 조합하여 길고 복잡한 표현식을 만들 수 있습니다. 앞으로도 기능을 추가할 예정이므로 필요한 기능이 없다면 알려 주십시오.

참고 Companion 5.0의 변경 사항

5.0에서는 표현식 엔진을 표준 JavaScript 해석기로 다시 구현하여 제어 흐름, 함수, 컬렉션 도우미를 추가했습니다. 고급 표현식을 참조하십시오. 이 과정에서 이전에 허용되던 일부 특수한 동작이 바뀌었습니다.

- 이제 문장을 줄바꿈이나 ;로 구분해야 합니다. 10 + 10 20 30처럼 구분자 없이 이어 쓴 문장은 더 이상 허용되지 않습니다.
- return과 반환값은 같은 줄에 있어야 합니다. return 바로 뒤에서 줄을 바꾸면 값을 반환하지 않고 표현식을 끝냅니다.
- 점을 이용한 객체 속성 접근($(custom:obj).name)이 이제 대괄호 접근($(custom:obj)['name'])과 동일하게 동작합니다. 이전에는 대괄호 접근만 값을 반환했습니다.

3.1 연산자

도움말

예제의 a, b, c는 사용자 변수, 모듈 변수, 숫자 리터럴 또는 다른 표현식으로 바꿀 수 있습니다.
여기서는 간단히 표시하기 위해 사용했습니다.

산술 연산자

연산자	구문	예제
덧셈	a + b	2 + 3의 결과는 5
뺄셈	a - b	5 - 2의 결과는 3
곱셈	a * b	4 * 3의 결과는 12
나눗셈	a / b	10 / 4의 결과는 2.5
거듭제곱	a ** b	2 ** 3의 결과는 8
나머지	a % b	7 % 3의 결과는 1(나머지)

주의

산술 연산자는 양쪽 값을 항상 숫자로 처리합니다. 특히 +는 **문자열을 합치는 데 사용할 수 없습니다**.
'a' + 'b'의 결과는 NaN입니다. 텍스트를 합치려면 템플릿 문자열이나 concat 함수를 사용하십시오.

비교 연산자

연산자	구문	예제
같음(느슨한 비교)	a == b	5 == '5'의 결과는 true
같음(엄격한 비교)	a === b	5 === '5'의 결과는 false
다름(느슨한 비교)	a != b	5 != '5'의 결과는 false
다름(엄격한 비교)	a !== b	5 !== '5'의 결과는 true
큼	a > b	5 > 3의 결과는 true
크거나 같음	a >= b	5 >= 5의 결과는 true
작음	a < b	3 < 5의 결과는 true
작거나 같음	a <= b	5 <= 3의 결과는 false

느슨한 비교(==)는 형식을 변환한 뒤 일치하면 같은 값으로 봅니다. 따라서 텍스트 '5'를 담은 변수와 숫자 5는 같습니다.
엄격한 비교(===)는 형식까지 같아야 합니다. 일부 연결은 변수 값을 텍스트로 제공하므로, 변수를 특정 값과 비교할 때는 보통 느슨한 비교가 적합합니다.

크기 비교 연산자(>, >=, <, <=)는 양쪽 값을 숫자로 변환한 뒤 비교합니다. 텍스트의 알파벳순을 비교하는 데는 사용할 수 없습니다.

논리 연산자

연산자	구문	예제
논리 AND	a && b	true && false의 결과는 false
논리 OR	a || b	0 || 'fallback'의 결과는 'fallback'
null 병합	a ?? b	undefined ?? 'fallback'의 결과는 'fallback'
논리 NOT	!a	!true의 결과는 false

||는 왼쪽 값이 거짓으로 평가되면(예: 0, 빈 문자열) 오른쪽 값을 사용합니다. ??는 왼쪽 값이 undefined 또는 null일 때만 오른쪽 값을 사용합니다. 값이 설정되지 않았을 수 있는 변수에 기본값을 주려면 ??를 사용하십시오.

```
$(custom:operator_name) ?? 'No operator'
```

삼항 연산자

조건(삼항) 연산자는 두 값 중 하나를 선택합니다.

연산자	구문	예제
조건	a ? b : c	$(custom:on_air) ? 'LIVE' : 'READY'

단항 연산자

연산자	구문	예제
부호 반전	-a	-(2 + 3)의 결과는 -5
숫자로 변환	+a	+'5'의 결과는 숫자 5
논리 NOT	!a	!0의 결과는 true
비트 NOT	~a	~5의 결과는 -6

비트 연산자

연산자	구문	예제
비트 AND	a & b	6 & 3의 결과는 2
비트 OR	a | b	6 | 3의 결과는 7
비트 XOR	a ^ b	6 ^ 3의 결과는 5
오른쪽 시프트	a >> b	8 >> 2의 결과는 2
왼쪽 시프트	a << b	2 << 2의 결과는 8

표현식 묶기

괄호로 부분 표현식을 묶어 계산 순서를 지정합니다.

```
(a + b) * c
```

연산자 우선순위는 일반적인 JavaScript 규칙을 따릅니다. 다만 여러 연산자를 함께 사용할 때는 우선순위에 의존하기보다 괄호를 넣는 편이 명확합니다.

객체와 배열

연산	구문
객체 정의	{ a: 1 }
배열 정의	[1, 2]
객체·배열 조회	$(my:var)['some-prop'] 또는 $(my:var)[0]
점으로 속성 접근	$(my:var).some_prop

임시 변수 대입

여러 줄 표현식에서는 중간 결과를 임시 변수에 저장할 수 있습니다.

연산자	구문
대입	a = 1
덧셈 후 대입	a += 1
뺄셈 후 대입	a -= 1
곱셈 후 대입	a *= 2
나눗셈 후 대입	a /= 2
나머지 대입	a %= 2
거듭제곱 후 대입	a **= 2
증가	a++ 또는 ++a
감소	a-- 또는 --a
논리 OR 후 대입	a ||= 1
논리 AND 후 대입	a &&= 1
null 병합 후 대입	a ??= 1
왼쪽 시프트 후 대입	a <<= 1
오른쪽 시프트 후 대입	a >>= 1
비트 XOR 후 대입	a ^= 1
비트 AND 후 대입	a &= 1
비트 OR 후 대입	a |= 1

다음 예제는 여러 줄에 걸쳐 값을 계산한 뒤 결과를 반환합니다.

```
total = $(custom:a)
total += $(custom:b)
total /= 2
total
```

3.2 함수

다양한 함수를 지원하며, 앞으로도 추가할 예정입니다. 필요한 함수가 없다면 알려 주십시오.

현재 지원하는 함수는 다음과 같습니다.

일반 연산

length(val)

전달한 항목의 길이를 구합니다.

- 문자열은 유니코드 문자소(grapheme) 수를 반환합니다.
- 배열은 요소 수를 반환합니다.
- JSON 등의 객체는 속성 수를 반환합니다.
- 숫자는 문자열로 표현했을 때의 길이를 반환합니다.

예: length("hello")는 5, length([5, 10, 15])는 3, length({a: 1, b: 2})는 2를 반환합니다.

숫자 연산

round(val)

가장 가까운 정수로 반올림합니다.

예: round(4.5)는 5, round(4.4)는 4를 반환합니다.

floor(val)

숫자를 정수로 내림합니다.

예: floor(4.9)는 4를 반환합니다.

ceil(val)

숫자를 정수로 올림합니다.

예: ceil(4.1)는 5를 반환합니다.

abs(val)

숫자의 절댓값을 구합니다. 음수이면 양수를 반환합니다.

예: abs(-4)와 abs(4)는 모두 4를 반환합니다.

fromRadix(val, [radix])

지정한 진법으로 표현된 문자열을 정수로 변환합니다. 진법의 기본값은 10입니다.

예: fromRadix("f", 16)은 15를 반환합니다.

toRadix(val, [radix])

정수를 2~36 범위에서 지정한 진법의 문자열로 변환합니다. 진법의 기본값은 10입니다.

예: toRadix(15, 16)은 "f"를 반환합니다.

toFixed(val, dp)

숫자를 지정한 소수 자릿수의 고정 정밀도 문자열로 변환합니다.

예: toFixed(3.14159, 2)는 "3.14", toFixed(3, 2)는 "3.00"을 반환합니다.

isNumber(val)

값이 숫자인지 확인합니다.

예: isNumber(5)와 isNumber("5")는 true, isNumber("five")는 false를 반환합니다.

max(val, val2, [val3, ...])

전달한 값 중 가장 큰 값을 구합니다.

예: max(3, 7, 5)는 7을 반환합니다. 값의 하한을 제한할 때 자주 사용합니다. max(0, $(custom:countdown))은 0보다 작아지지 않습니다.

min(val, val2, [val3, ...])

전달한 값 중 가장 작은 값을 구합니다.

예: min(3, 7, 5)는 3을 반환합니다. max와 함께 사용하면 값의 범위를 제한할 수 있습니다.
min(100, max(0, $(custom:level)))은 값을 0~100으로 제한합니다.

randomInt([min], [max])

지정한 범위의 양 끝값을 포함하여 임의의 정수를 생성합니다. 범위를 지정하지 않으면 0~10을 사용합니다.

예: randomInt(1, 6)은 주사위처럼 1~6의 값을 반환합니다.

log(v, [base])

숫자의 로그를 계산합니다. 밑을 생략하면 자연로그(밑 e)를, 지정하면 해당 밑의 로그를 반환합니다.

예: log(8, 2)와 log(1000, 10)은 각각 3, 3을 반환합니다.

log10(v)

숫자의 상용로그(밑 10)를 계산합니다.

예: log10(1000)은 3을 반환합니다.

exp(v)

e의 거듭제곱을 계산합니다. 자연로그의 역함수입니다.

예: exp(0)은 1을 반환합니다. e 자체의 값은 exp(1)입니다.

sqrt(v)

숫자의 제곱근을 계산합니다.

예: sqrt(9)는 3을 반환합니다.

cbrt(v)

숫자의 세제곱근을 계산합니다.

예: cbrt(8)은 2를 반환합니다.

pow(base, exponent)

밑(base)을 지수(exponent)만큼 거듭제곱합니다. ** 연산자와 같은 기능이지만, 의미를 명확히 표현하거나 다른 언어에 익숙한 사용자가 활용할 수 있도록 함수로도 제공합니다.

예: pow(2, 10)은 1024를 반환합니다.

sin(v)

라디안 단위의 각도를 받아 사인값을 계산합니다.

예: sin(0)은 0을 반환합니다.

cos(v)

라디안 단위의 각도를 받아 코사인값을 계산합니다.

예: cos(0)은 1을 반환합니다.

tan(v)

라디안 단위의 각도를 받아 탄젠트값을 계산합니다.

예: tan(0)은 0을 반환합니다.

asin(v)

숫자의 아크사인 각도를 라디안으로 계산합니다.

예: asin(0)은 0을 반환합니다.

acos(v)

숫자의 아크코사인 각도를 라디안으로 계산합니다.

예: acos(0)은 PI / 2를 반환합니다.

atan(v)

숫자의 아크탄젠트 각도를 라디안으로 계산합니다.

예: atan(0)은 0을 반환합니다.

asinh(v)

숫자의 역쌍곡사인을 계산합니다.

예: asinh(0)은 0을 반환합니다.

acosh(v)

숫자의 역쌍곡코사인을 계산합니다.

예: acosh(1)은 0을 반환합니다.

atanh(v)

숫자의 역쌍곡탄젠트를 계산합니다.

예: atanh(0)은 0을 반환합니다.

2 * PI처럼 상수 PI를 값으로 사용할 수도 있습니다.

문자열 연산

trim(val)

문자열 앞뒤의 공백을 제거합니다.

예: trim(" hello ")은 "hello"를 반환합니다.

strlen(val)

문자열의 길이를 구합니다. 유니코드 문자열은 문자소가 아닌 코드 단위로 세므로, 이모지와 일부 문자는 둘 이상으로 계산됩니다.

substr(val, indexStart, indexEnd)

substr()는 indexStart부터 indexEnd 직전까지의 문자를 추출합니다. 유니코드 문자열은 문자소가 아닌 코드 단위를 기준으로 합니다.

예: substr("Companion", 0, 3)은 "Com", substr("Companion", -3)은 "ion"을 반환합니다.

- indexStart >= str.length이면 빈 문자열을 반환합니다.
- indexStart < 0이면 문자열 끝에서부터 위치를 계산합니다. 정확한 시작 위치는 max(indexStart + str.length, 0)입니다.
- indexStart를 생략하거나 undefined이거나 숫자로 변환할 수 없으면 0으로 처리합니다.
- indexEnd를 생략하거나 undefined이거나 숫자로 변환할 수 없거나 indexEnd >= str.length이면, substr()는 문자열 끝까지 추출합니다.
- indexEnd < 0이면 문자열 끝에서부터 위치를 계산합니다.
- 음수 값을 정규화한 뒤 indexEnd <= indexStart이면 빈 문자열을 반환합니다.

도움말

음수일 때의 동작을 원하지 않으면 max(0, index)로 값을 0 이상으로 제한할 수 있습니다.

split(str, separator)

구분자를 기준으로 문자열을 나눕니다.

예: split("1:30:45", ":")는 ["1", "30", "45"]를 반환하므로, split($(internal:time_hms), ":")[0]은 현재 시를 반환합니다.

join(arr, separator)

배열의 값 사이에 지정한 구분자를 넣어 하나의 문자열로 합칩니다.

예: join([1, 30, 45], ":")는 "1:30:45"를 반환합니다.

concat(str1, [str2, ...])

하나 이상의 값을 하나의 문자열로 합칩니다.

예: 변수 값이 2일 때 concat("Cam ", $(custom:camera))는 "Cam 2"를 반환합니다. + 연산자는 숫자 덧셈만 수행하므로 텍스트를 합칠 때는 이 함수나 템플릿 문자열을 사용합니다.

includes(val, find)

문자열에 특정 값이 포함되어 있는지 확인합니다.

예: includes("Companion is great!", "great")는 true를 반환합니다.

indexOf(val, find, offset)

문자열에서 해당 값이 처음 나타나는 위치를 구합니다. 유니코드 문자열은 문자소가 아닌 코드 단위를 기준으로 합니다.

검색을 시작할 offset을 지정할 수 있으며, 생략하면 0번째 위치인 문자열 처음부터 검색합니다.

값을 찾지 못하면 -1을, 찾으면 처음 나타나는 위치의 인덱스를 반환합니다.

lastIndexOf(val, find, offset)

문자열 끝에서부터 검색하여 해당 값이 마지막으로 나타나는 위치를 구합니다. 유니코드 문자열은 문자소가 아닌 코드 단위를 기준으로 합니다.

끝에서부터 검색을 시작할 offset을 선택적으로 지정할 수 있습니다.

값을 찾지 못하면 -1을, 찾으면 마지막으로 나타나는 위치의 인덱스를 반환합니다. 문자열 처음의 위치는 0입니다.

toUpperCase(val)

문자열의 모든 문자를 대문자로 변환합니다.

예: 변수 값이 Interview일 때 toUpperCase($(custom:scene_name))은 "INTERVIEW"를 반환합니다.

toLowerCase(val)

문자열의 모든 문자를 소문자로 변환합니다.

예: toLowerCase("Interview")는 "interview"를 반환합니다.

replaceAll(val, find, replace)

문자열에서 특정 값을 찾아 모든 일치 항목을 새 문자열로 바꿉니다.

예: replaceAll("This is great!", "This", "Companion")은 Companion is great!를 반환합니다.

stringCompare(a, b)

지역별 언어 규칙에 따라 두 문자열의 정렬 순서를 비교합니다. a가 b보다 앞이면 음수, 같으면 0, a가 b보다 뒤이면 양수를 반환합니다.

문자열이나 객체 배열을 정렬할 때 arraySort의 비교 함수로 특히 유용합니다.

예: arraySort(["banana", "apple", "cherry"], (a, b) => stringCompare(a, b))는 ["apple", "banana", "cherry"]를 반환합니다.

encode(str, enc)

문자열을 지정한 형식('hex','base64')으로 인코딩합니다. enc를 생략하면 latin1을 사용합니다.

예: encode("Companion","hex")는 "436f6d70616e696f6e"를 반환합니다.

decode(str, enc)

지정한 형식('hex','base64')으로 인코딩된 문자열을 디코딩합니다. enc를 생략하면 latin1을 사용합니다.

예: decode("436f6d70616e696f6e","hex")는 "Companion"을 반환합니다.

encodeURI(str)

문자열을 유효한 URI(Uniform Resource Identifier)로 인코딩합니다.

예: encodeURI('hello world&1')은 "hello%20world&1"을 반환합니다.

decodeURI(str)

인코딩된 URI(Uniform Resource Identifier)를 인코딩 전 문자열로 되돌립니다.

예: decodeURI('hello%20world&1')는 "hello world&1"을 반환합니다.

encodeURIComponent(str)

문자열을 유효한 URI(Uniform Resource Identifier)의 구성 요소로 인코딩합니다.

예: encodeURIComponent('hello world&1')는 "hello%20world%261"을 반환합니다.

decodeURIComponent(str)

인코딩된 URI(Uniform Resource Identifier) 구성 요소를 인코딩 전 문자열로 되돌립니다.

예: decodeURIComponent('hello%20world%261')는 "hello world&1"을 반환합니다.

색상 연산

색상을 만들고, 확인하고, 조절하는 함수입니다. 색상을 받는 곳에는 CSS 색상
문자열("#f00", "rgb(255 0 0)", "hsl(0, 100%, 50%)" 또는 "red" 같은 CSS 색상 키워드),
색상 선택기가 생성하는 24비트 0xRRGGBB 숫자, colorTo* 함수가 반환하는 채널 객체를 사용할 수 있습니다.
색상으로 해석할 수 없는 값이면 null을 반환합니다.

rgb(r, g, b, ?a) / rgba(r, g, b, ?a)

빨강·초록·파랑 채널(각각 0~255)과 선택적인 알파값(0~1)으로 색상을 만듭니다. rgb()/rgba() 문자열을 반환합니다.

채널 대신 색상 인수 하나를 전달하면 해당 색상을 rgb() 형식으로 바꿉니다. 예를 들어 rgb("#336699")는
"rgb(51, 102, 153)"을 반환합니다.

예: rgb(255, 0, 0)은 "rgb(255, 0, 0)", rgb(255, 0, 0, 0.5)는 "rgba(255, 0, 0, 0.5)"를 반환합니다.

hsl(h, s, l, ?a) / hsla(h, s, l, ?a)

색조(0~360), 채도·명도(각각 0~100), 선택적인 알파값(0~1)으로 색상을 만듭니다. hsl()/hsla() 문자열을 반환합니다.
색상 인수 하나를 전달하면 해당 색상을 hsl() 형식으로 바꿉니다.

예: hsl(0, 100, 50)은 "hsl(0, 100%, 50%)"를 반환합니다.

hsv(h, s, v, ?a)

색조(0~360), 채도·밝기(각각 0~100), 선택적인 알파값(0~1)으로 색상을 만듭니다. rgb()/rgba() 문자열을 반환합니다.
색상 인수 하나를 전달하면 해당 색상을 변환합니다.

예: hsv(0, 100, 100)은 "rgb(255, 0, 0)"을 반환합니다.

colorToRgb(color) / colorToHsl(color) / colorToHsv(color)

색상을 채널별 객체로 나눕니다. 예를 들어 colorToRgb("#ff0000")은 { r: 255, g: 0, b: 0, a: 1 }, colorToHsl("#ff0000")은 { h: 0, s: 100, l: 50, a: 1 }을 반환합니다.

colorToHex(color)

색상을 "#rrggbb" 형식의 16진수 문자열로 변환합니다. 예를 들어 colorToHex("rgb(255, 0, 0)")는 "#ff0000"을 반환합니다.

isColor(value)

이 함수들이 해석할 수 있는 색상 값인지 확인합니다. 예를 들어 isColor("#ff0000")은 true, isColor("notacolor")는 false를 반환합니다.

colorAlpha(color, alpha)

색상의 알파값(0~1)을 설정합니다. rgb()/rgba() 문자열을 반환합니다.

예: colorAlpha("#ff0000", 0.5)는 "rgba(255, 0, 0, 0.5)"를 반환합니다.

colorLighten(color, ?amount) / colorDarken(color, ?amount)

지정한 양(0~1, 기본값 0.1)만큼 색상을 밝게 또는 어둡게 만듭니다. rgb()/rgba() 문자열을 반환합니다.

예: colorLighten("#808080", 0.2)는 더 밝은 회색을 반환합니다.

colorSaturate(color, ?amount)

지정한 양(0~1, 기본값 0.1)만큼 채도를 높입니다. 음수를 지정하면 채도를 낮춥니다. rgb()/rgba() 문자열을 반환합니다.

colorInvert(color)

색상을 반전합니다. rgb()/rgba() 문자열을 반환합니다. 예를 들어 colorInvert("#000000")는 "rgb(255, 255, 255)"를 반환합니다.

colorMix(colorA, colorB, ?ratio)

CIE Lab 색 공간에서 두 색을 혼합합니다. ratio는 두 번째 색의 비율(0~1, 기본값 0.5)입니다.
rgb()/rgba() 문자열을 반환합니다.

예: colorMix("#ff0000", "#0000ff")는 빨강과 파랑의 중간색을 반환합니다.

colorIsDark(color)

밝은 글자를 올려야 할 만큼 색상이 어두운지 확인합니다. 불리언값을 반환합니다.

예: colorIsDark("#000000")은 true, colorIsDark("#ffffff")는 false를 반환합니다.

변수 연산

parseVariables(string, ?undefinedValue)

$(custom:$(custom:b))처럼 중첩된 변수 평가가 필요한 경우가 있습니다. 표현식 해석기는 이 중첩 구문을 직접 지원하지 않습니다. 표현식 안에서 중첩 변수를 평가하려면 문자열을 parseVariables에 전달하십시오. 문자열 변수와 템플릿 구문을 해석합니다.

변수 값이 undefined일 때 대신 사용할 값을 선택적으로 지정할 수 있습니다. 대체 값은 문자열이어야 합니다.

예: parseVariables('$(custom:$(custom:b))')

getVariable(string) or getVariable(string, string)

변수 값을 동적으로 가져옵니다.

참고: 가능하면 $(internal:time_hms) 구문을 사용하십시오. Companion이 변수 사용을 정적으로 감지하여 필요할 때 참조 이름을 자동으로 바꿀 수 있습니다.

예: getVariable('internal:time_hms') 또는 getVariable('internal', 'time_hms')

blink(number, ?number)

켜짐과 꺼짐을 번갈아 나타내는 0/1 값을 생성합니다. 첫 번째 숫자는 주기 길이(밀리초)이며 최소값은 100입니다.
두 번째 매개변수는 켜짐 상태의 시간 비율을 나타내는 0~1의 선택값이며, 기본값은 0.5입니다.

예: blink(1000)은 1초마다 깜박이며, 500ms 동안 켜지고 500ms 동안 꺼집니다. blink(1000, 0.25)는 1초마다 깜박이며, 250ms 동안 켜지고 750ms 동안 꺼집니다.

도움말 팁

- 이 함수가 반환하는 0/1은 별도 변환 없이 불리언값으로 사용할 수 있습니다.
- 초당 깜박임 횟수로 지정하려면 첫 번째 매개변수에 1000/n을 사용합니다. 예를 들어 초당 세 번 깜박이려면 blink(1000/3)을 사용합니다.

oscillate(period, ?waveform, ?phase)

0과 1 사이를 부드럽게 반복하는 값을 생성합니다. 요소의 불투명도를 맥동시키거나, 게이지를 움직이거나, 색상을 서서히 바꾸는 등 버튼 애니메이션에 유용합니다.

- period는 한 주기의 길이(밀리초)이며 최소값은 100입니다.
- waveform은 주기의 파형입니다.
  - 'sine'(기본값): 0에서 시작해 주기의 중간에 1까지 부드럽게 올라간 뒤 다시 0으로 내려갑니다.
  - 'triangle': 0에서 1까지 일정하게 올라갔다가 다시 내려갑니다.
  - 'sawtooth': 0에서 1까지 일정하게 올라간 뒤 즉시 0으로 돌아갑니다.
  - 'square': 주기의 앞 절반은 1, 뒤 절반은 0입니다.
- phase는 주기를 0~1의 비율만큼 이동합니다. 여러 진동값을 서로 다른 시점에 움직이게 할 때 사용합니다. 예를 들어 phase가 0.5이면 위상을 지정하지 않은 진동값이 0일 때 이 값은 1입니다.

예: 요소의 **불투명도(Opacity)**에 oscillate(2000) * 100을 사용하면 2초마다 서서히 나타났다가 사라집니다.

예: oscillate(1000, 'sawtooth')는 1초마다 0에서 1로 올라갑니다.

주기가 같은 진동값은 서로 다른 버튼에서도 동일한 시점에 움직입니다.

참고

oscillate는 초당 10회 갱신되므로 주기가 매우 짧으면 움직임이 끊겨 보입니다. 버튼 요소의 속성, 피드백, 로컬 변수·표현식 변수, 트리거 조건에 사용할 수 있습니다. 액션은 지속적으로 갱신되지 않고 특정 시점에 한 번 실행되므로 액션 옵션에는 사용할 수 없습니다.

불리언 연산

bool(val)

값을 불리언으로 변환합니다.

다음 값은 참으로 해석합니다.

- 0이 아닌 숫자
- "false"와 "0"을 제외한 비어 있지 않은 문자열

나머지 값(0, "", "false", "0", false, null, undefined)은 거짓입니다.

텍스트로 값을 저장하는 변수에 특히 유용합니다. 일반 조건에서 문자열 "false"는 참으로 처리되지만, bool($(obs:streaming))은 의도한 결과를 반환합니다.

객체·배열 연산

jsonpath(obj, path)

객체나 배열에서 JSONPath로 값을 찾습니다. 복잡한 데이터를 하나의 변수로 제공하는 연결에서 값을 추출할 때 유용합니다.

입력은 객체 또는 문자열로 변환된 객체일 수 있으며, 출력은 입력과 같은 형식입니다.

예를 들어 변수에 { "scenes": [{ "name": "Camera 1" }, { "name": "Slides" }] }가 들어 있다면 다음과 같습니다.

- jsonpath($(custom:state), '$.scenes[0].name')는 "Camera 1"을 반환합니다.
- jsonpath($(custom:state), '$.scenes[*].name')는 ["Camera 1", "Slides"]를 반환합니다.

더 많은 사용 예제는 https://jsonpath.com/ 에서 확인할 수 있습니다.

jsonparse(str)

JSON 문자열을 해석하여 객체로 만듭니다.

유효하지 않은 입력이면 오류를 발생시키는 대신 null을 반환합니다.

예: jsonparse('{"a":1}')의 결과는 객체 { a: 1 }입니다.

jsonstringify(obj)

객체를 JSON 문자열로 변환합니다.

유효하지 않은 입력이면 오류를 발생시키는 대신 null을 반환합니다.

예: jsonstringify({ a: 1 })의 결과는 {"a":1}을 담은 문자열입니다.

arrayIncludes(arr, val)

배열에 해당 값이 포함되어 있는지 확인합니다.

유효하지 않은 입력이면 false를 반환합니다.

arrayIndexOf(arr, val, [offset])

배열에서 해당 값이 처음 나타나는 위치를 구합니다.

검색을 시작할 offset을 지정할 수 있으며, 생략하면 0번째 위치인 배열 처음부터 검색합니다.

값을 찾지 못하면 -1을, 찾으면 처음 나타나는 위치의 인덱스를 반환합니다.

arrayLastIndexOf(arr, val, offset)

배열 끝에서부터 검색하여 해당 값이 마지막으로 나타나는 위치를 구합니다.

끝에서부터 검색을 시작할 offset을 선택적으로 지정할 수 있습니다.

값을 찾지 못하면 -1을, 찾으면 마지막으로 나타나는 위치의 인덱스를 반환합니다. 배열 처음의 위치는 0입니다.

arraySlice(arr, start, end)

원본 배열을 바꾸지 않고 일부를 얕게 복사하여 반환합니다.

start는 시작 인덱스(0은 처음), end는 추출에서 제외할 끝 인덱스입니다. 둘 다 생략할 수 있으며,
음수는 끝에서부터 계산합니다.

arraySlice([1, 2, 3, 4, 5], 1, 3)은 [2, 3]을 반환합니다.

유효하지 않은 입력이면 undefined를 반환합니다.

arrayConcat(...arrays)

여러 배열을 하나의 새 배열로 합칩니다. 배열이 아닌 인수는 요소 하나로 묶어 추가합니다.

arrayConcat([1, 2], [3, 4], 5)는 [1, 2, 3, 4, 5]를 반환합니다.

arrayFlat(arr)

중첩 배열을 한 단계 평탄화합니다. arrayFlat([1, [2, 3], [4, [5]]])는 [1, 2, 3, 4, [5]]를 반환합니다.

유효하지 않은 입력이면 undefined를 반환합니다.

배열 반복 연산

배열의 각 요소에 함수(보통 x => ... 형태의 화살표 함수)를 적용합니다. 대부분의 콜백은 (element, index)를 받고,
arrayReduce는 (accumulator, element, index)를 받습니다. 화살표 함수와 함께 사용하면 더욱 다양하게
활용할 수 있습니다. 고급 표현식을 참조하십시오.

arrayMap(arr, fn)

모든 요소에 fn을 적용한 새 배열을 반환합니다. arrayMap([1, 2, 3], x => x * 2)는 [2, 4, 6]을 반환합니다.

arrayFilter(arr, fn)

fn이 true를 반환하는 요소만 담은 새 배열을 반환합니다. arrayFilter([1, 2, 3, 4], x => x % 2 === 0)은 [2, 4]를 반환합니다.

arrayReduce(arr, fn, initial)

모든 요소를 하나의 값으로 합칩니다. fn은 현재까지의 누적값과 현재 요소를 받습니다.
arrayReduce([1, 2, 3, 4], (sum, x) => sum + x, 0)은 10을 반환합니다.

arrayForEach(arr, fn)

값을 누적하는 등의 부수 효과를 위해 각 요소에 fn을 실행합니다. 반환값은 없습니다.

arrayFind(arr, fn) / arrayFindIndex(arr, fn)

fn이 true를 반환하는 첫 요소 또는 그 인덱스를 반환합니다. 일치하는 요소가 없으면 각각 undefined / -1을 반환합니다.

arraySome(arr, fn) / arrayEvery(arr, fn)

fn이 한 요소 이상에서 true를 반환하는지(arraySome), 모든 요소에서 true를 반환하는지(arrayEvery)를 확인합니다.

arraySort(arr, fn)

원본을 바꾸지 않고 정렬된 배열의 복사본을 반환합니다. 비교 함수를 생략하면 요소를 문자열로 정렬합니다. 순서를 지정하려면 음수·0·양수를 반환하는 fn(a, b)를 전달하십시오. arraySort([3, 1, 2], (a, b) => a - b)는 [1, 2, 3]을 반환합니다.

문자열을 정렬하거나 객체를 문자열 속성으로 정렬하려면 stringCompare를 비교 함수로 사용합니다.
예: arraySort($(custom:names), (a, b) => stringCompare(a, b)). 자세한 내용은 stringCompare를 참조하십시오.

arrayReverse(arr)

요소 순서를 뒤집은 배열의 복사본을 반환합니다.

objectKeys(obj) / objectValues(obj)

객체 또는 배열의 키나 값을 배열로 반환합니다.

시간 연산

unixNow()

현재 Unix 시간을 밀리초로 구합니다.

timestampToSeconds(timestamp)

'HH:MM:SS' 형식의 타임스탬프를 해당 초 수로 변환합니다.

예: 00:10:15는 615가 됩니다.

반대 변환에는 secondsToTimestamp(str)을 사용합니다.

secondsToTimestamp(seconds, format) / msToTimestamp(milliseconds, format)

초 또는 밀리초 수를 지정한 형식의 타임스탬프 문자열로 변환합니다. 기본 형식은 초의 경우 'nHH:mm:ss',
밀리초의 경우 'nmm:ss.S'입니다.

format 매개변수를 지정하여 출력 문자열에 포함할 구성 요소를 선택할 수 있습니다.

사용 가능한 구성 요소는 다음과 같습니다.

- n: 음수 부호
- dd: 일

- HH / hh: 24시간제 / 12시간제
- mm: 분
- ss: 초
- S / SS / SSS: 소수 초 표시. 정밀도는 각각 1/10초, 1/100초, 밀리초입니다. msToTimestamp와 소수 초를 입력한 secondsToTimestamp에서 사용할 수 있습니다.
- a: AM/PM 표시

문자를 반복하여 자릿수를 지정합니다. 예를 들어 HHH는 시를 3자리로, ssss는 초를 4자리로 표시합니다.
밀리초(S)는 1~3자리를 지원합니다.

예: secondsToTimestamp(685, 'mm:ss')는 "11:25"를 반환하며,
secondsToTimestamp($(custom:remaining), 'HH:mm:ss')는 카운트다운 변수를 시계 형식으로 표시합니다.

기본적으로 각 시간 단위는 나머지 값을 표시합니다. 예를 들어 시는 24로, 분은 60으로 나눈 나머지를 표시합니다.
누적값을 표시하려면 단위를 대괄호 [unit]로 감싸십시오. [HH]:mm:ss는 시를 24로 나눈 나머지 대신 누적 시간으로, [mm]:ss는 분을 60으로 나눈 나머지 대신 누적 분으로 표시합니다. 형식 문자열마다 가장 큰 단위는 하나만 지정할 수 있으며, H, h, m, s 중에서 선택해야 합니다.

timeOffset(timestamp, offset, 12hour)

전달한 타임스탬프(hours:minutes 또는 hours:minutes:seconds)에 시·분·초 단위의 차이를 적용합니다.
선택적으로 12시간 형식으로 반환할 수 있습니다.

예: timeOffset($(internal:time_hms), -5)는 현재보다 5시간 전의 시·분·초를 반환합니다.

offset에는 타임스탬프도 사용할 수 있습니다. timeOffset($(internal:time_hms), "01:30:00", true)는 현재 시간에 1시간 30분을 더한 뒤 해당 시간을 12시간 형식으로 반환합니다.

timeDiff(fromTime, toTime)

두 타임스탬프 사이의 초 수를 반환합니다. hours:minutes, hours:minutes:seconds,
YYYY-MM-DDTHH:mm:ss.sssZ 형식을 지원합니다.

예: timeDiff($(internal:time_hms), "18:00:00")는 같은 날의 18:00:00까지 남은 초 수를 반환하며, 해당 시간이 지나면 음수를 반환합니다. 날짜·시간 문자열을 사용한 예인 timeDiff($(internal:time_hms), "2024-07-04T20:00-04:00")는 현재 Companion 시간부터 UTC-4 시간대의 2024년 7월 4일 오후 8시까지의 초 수를 반환합니다.

반환된 초 수를 secondsToTimestamp에 전달하여 원하는 형식으로 표시할 수도 있습니다.

날짜 연산

parseDate(value)

날짜 값을 해석하여 Unix 타임스탬프를 밀리초로 반환합니다. 다음 입력을 받습니다.

- 숫자: Unix 밀리초로 처리합니다.
- 문자열: JavaScript의 Date() 생성자로 해석합니다. 정확한 결과를 위해 ISO 8601 형식(예: "2024-06-15T14:30:00Z")을 권장합니다. MM/DD/YYYY처럼 모호한 형식은 결과를 보장하지 않습니다.

값을 해석할 수 없으면 null을 반환합니다.

예: parseDate("2024-06-15T12:00:00Z")는 해당 날짜의 Unix 밀리초를 반환합니다.

반환값은 unixNow()와 같은 Unix 밀리초 형식이므로 다른 모든 날짜 함수에 사용할 수 있습니다.

dateYear(value, timezone?), dateMonth(value, timezone?), dateDay(value, timezone?)

날짜 값(Unix 밀리초 또는 날짜 문자열)에서 날짜 구성 요소를 추출합니다.

- dateYear는 전체 연도(예: 2024)를 반환합니다.
- dateMonth는 1(1월)~12(12월)를 반환합니다.
- dateDay는 월의 날짜인 1~31을 반환합니다.

예: dateYear(unixNow())는 현재 연도를, dateMonth(unixNow(), 'UTC')는 UTC 기준 현재 월을 반환합니다.

dateHour(value, timezone?), dateMinute(value, timezone?), dateSecond(value, timezone?)

날짜 값에서 시간 구성 요소를 추출합니다.

- dateHour는 0~23을 반환합니다.
- dateMinute는 0~59를 반환합니다.
- dateSecond는 0~59를 반환합니다.

예: dateHour(unixNow())는 현지 시간의 현재 시를, dateHour(unixNow(), 'America/New_York')는 뉴욕의 현재 시를 반환합니다.

dateWeekday(value, timezone?)

요일을 숫자로 반환합니다. 0 = 일요일, 1 = 월요일, ..., 6 = 토요일입니다.

예: dateWeekday(unixNow())는 오늘의 요일 번호를 반환합니다.

dateFormat(value, formatString, timezone?, locale?)

날짜 값을 사용자 지정 문자열로 표시합니다. 형식 문자열에는 dayjs 호환 토큰을 사용합니다.

토큰	설명	예제
YYYY / YY	연도	2024 / 24
MMMM / MMM / MM / M	월	June / Jun / 06 / 6
dddd / ddd / DD / D	요일 / 월의 날짜	Saturday / Sat / 05 / 5
HH / H	24시간제의 시	09 / 9
hh / h	12시간제의 시	09 / 9
mm / m	분	05 / 5
ss / s	초	03 / 3
SSS	밀리초	123
A / a	AM/PM	AM / am

형식에 'iso'를 전달하면 항상 UTC인 ISO 8601 문자열을 얻습니다.
dateFormat(value, 'iso')는 "2024-06-15T14:30:00.000Z"를 반환합니다.

예: dateFormat(unixNow(), 'YYYY-MM-DD HH:mm:ss')는 "2024-06-15 14:30:00" 같은 문자열을 반환합니다.

예: dateFormat(unixNow(), 'dddd, MMMM D, YYYY')는 "Saturday, June 15, 2024" 같은 문자열을 반환합니다.

월(MMMM / MMM)과 요일(dddd / ddd) 이름에는 언어 설정이 적용됩니다. 기본으로 시스템 로캘을 따르며 internal:date_weekday 변수와 같습니다. 따라서 프랑스어 시스템에서는 위 예제가 "samedi, juin 15, 2024"를 반환합니다. 네 번째 인수에 BCP 47 로캘을 명시하면 이를 덮어쓸 수 있습니다. 예를 들어 dateFormat(unixNow(), 'dddd', 'UTC', 'fr')는 "samedi"를 반환하고, dateFormat(unixNow(), 'dddd', 'UTC', 'en')은 시스템 로캘과 관계없이 영어를 사용합니다.

dateAdd(value, amount, unit)

날짜 값에 기간을 더하거나 뺍니다. 결과를 Unix 타임스탬프(밀리초)로 반환합니다.

지원 단위는 seconds, minutes, hours, days, weeks, months, years이며 단수형과 복수형 모두 사용할 수 있습니다.

시간을 빼려면 음수를 지정합니다.

예: dateAdd(unixNow(), 7, 'days')는 지금부터 7일 후의 Unix 밀리초를 반환합니다.

예: dateFormat(dateAdd(unixNow(), 1, 'months'), 'YYYY-MM-DD')는 지금부터 1개월 후의 날짜를 지정한 형식으로 표시합니다.

참고: 월을 더한 결과의 날짜가 해당 월의 일수를 넘으면 다음 달로 넘어갑니다. 예를 들어 1월 31일에 1개월을 더하면 2월이 아닌 3월이 됩니다.

도움말

날짜 구성 요소 함수와 dateFormat에는 IANA 시간대 문자열(예: 'UTC', 'Europe/London', 'America/New_York')을 선택적으로 지정할 수 있습니다. 생략하면 설정의 시간대를 사용합니다. 모든 날짜 함수는 잘못된 입력에 null을 반환하며, dateFormat은 ''를 반환합니다.

3.3 고급 표현식

기본 표현식에서는 대부분의 버튼 텍스트와 피드백에 충분한 값, 연산자, 함수, 템플릿을 다룹니다.
더 복잡한 로직을 위해 표현식은 값을 바꿀 수 있는 변수, 제어문(if, 반복문), 사용자 정의 함수도 지원합니다.
사용자 변수나 표현식 변수의 값을 계산하는 등 긴 여러 줄 표현식에서 특히 유용합니다.

이 절은 기본 내용을 바탕으로 하므로 먼저 연산자, 함수, 템플릿 문자열에 익숙해지는 것이 좋습니다.

변수와 유효 범위

긴 표현식에서는 중간값을 저장하고 갱신해야 할 때가 많습니다. 변수는 두 가지 방법으로 만들 수 있습니다.

- **단순 대입**: total = 0은 변수 값을 설정하며, 필요하면 변수를 만듭니다.
- **선언**: let total = 0 또는 const total = 0은 새 변수를 명시적으로 만듭니다. const에는 이후 다른 값을 대입할 수 없습니다.

```
let count = 0
count = count + 1
count // 1
```

설정한 적이 없는 변수를 읽으면 존재하지 않는 $() 변수와 마찬가지로 undefined가 됩니다.

유효 범위

유효 범위(scope)는 변수가 존재하는 영역입니다. if, for, while의 { ... } 본문과 함수 본문 등 모든 블록은 새 유효 범위를 만듭니다. 블록 안에서 선언한 변수는 그 안에서만 존재합니다.

변수를 읽거나 값을 대입할 때 Companion은 현재 범위를 먼저 확인하고, 바깥 범위를 차례로 확인합니다.
이 규칙은 다음 세 가지 동작으로 이어집니다.

1. 단순 대입은 가장 가까운 기존 변수를 갱신합니다. 따라서 반복문에서 값을 누적할 수 있습니다.
반복문 안의 대입이 앞서 선언한 바깥쪽 total을 갱신합니다.

```
let total = 0
for (const x of [1, 2, 3]) {
  total = total + x // updates the outer `total`
}
total // 6
```

2. let / const는 항상 현재 범위에 새 변수를 만듭니다. 바깥에 같은 이름의 변수가 있어도 마찬가지입니다.
이를 가리기(shadowing)라고 합니다. 안쪽 변수는 자신의 블록 안에서만 바깥 변수를 가립니다.

```
let name = 'outer'
if (true) {
  let name = 'inner' // a separate variable
  name // 'inner' here
}
name // still 'outer'
```

3. 어디에도 존재하지 않는 이름에 단순 대입하면 현재 범위에 변수를 만듭니다. 따라서 블록 안에서 처음 설정한 변수는 블록이 끝나면 사라집니다.

```
if (true) {
  temp = 5 // created inside the block
}
temp // gone here -> undefined
```

반복문이나 if가 끝난 뒤에도 값이 남아 있어야 한다면 위의 반복문 예제처럼 블록 앞에서 선언한 뒤 블록 안에서 대입하십시오.

도움말

새 변수가 필요하거나 바깥 변수와의 충돌을 피하려면 let/const를 사용합니다. 누적 합계처럼 바깥 범위의 기존 변수를 갱신하려면 name = ... 형태의 단순 대입을 사용합니다.

const는 재대입할 수 없으므로 바꾸지 않을 값을 보호할 때 유용합니다.

```
const limit = 10
limit = 20 // error - the expression is invalid
```

제어 흐름

if / else

if는 조건이 참일 때 블록을 실행합니다. 필요하면 else if와 else 분기를 추가할 수 있습니다.

```
let label = ''
if ($(custom:level) > 90) {
  label = 'HIGH'
} else if ($(custom:level) > 50) {
  label = 'MEDIUM'
} else {
  label = 'LOW'
}
label
```

조건은 0, 빈 문자열, null, undefined, NaN이 아니면 참으로 처리됩니다. 삼항 연산자 ?:와 같은 규칙입니다.
변수에 텍스트 "false"나 "0"이 들어 있는 경우에는 bool로 감싸십시오.

반복문

for...of는 배열 요소를 순회하고, for는 횟수를 세며 반복하고, while은 조건이 유지되는 동안 반복합니다.

```
let total = 0
for (const item of $(custom:cart)) {
  total = total + item.price * item.qty
}
total
```

```
let sum = 0
for (let i = 1; i <= 10; i++) {
  sum = sum + i
}
sum // 55
```

break는 반복문을 일찍 끝내고, continue는 다음 반복으로 넘어갑니다.

```
let firstBig = -1
for (const x of $(custom:values)) {
  if (x > 100) {
    firstBig = x
    break
  }
}
firstBig
```

결과 반환

if나 for 같은 문장은 그 자체로 값을 만들지 않습니다. 값을 반환하려면 마지막에 표현식(보통 값을 누적한 변수)을 놓거나 return을 사용합니다.

```
let total = 0
for (const x of [1, 2, 3]) {
  total = total + x
}
total // <- the result
```

함수

(parameters) => result 형태의 화살표 구문으로 함수를 정의하고 이름으로 호출할 수 있습니다.

```
const double = (x) => x * 2
double(21) // 42
```

함수는 정의된 범위의 변수를 사용할 수 있으며, 자기 자신을 재귀적으로 호출할 수도 있습니다.

```
const factor = 3
const scale = (x) => x * factor // remembers `factor`
scale(10) // 30
```

```
let fib = (n) => (n < 2 ? n : fib(n - 1) + fib(n - 2))
fib(10) // 55
```

문장이 여러 개라면 함수 본문을 블록으로 만들고 return을 사용합니다.

```
let clamp = (value, lowest, highest) => {
  if (value < lowest) { return lowest }
  if (value > highest) { return highest }
  return value
}
clamp($(custom:level), 0, 100)
```

컬렉션 함수

배열의 각 요소에 함수를 적용하는 컬렉션 도우미와 함께 사용하면 더욱 다양하게 활용할 수 있습니다. arrayMap, arrayFilter, arrayReduce, arrayFind 등의 전체 목록은 배열 반복 함수를 참조하십시오.

```
// total price of everything in the cart
arrayReduce($(custom:cart), (sum, item) => sum + item.price * item.qty, 0)
```

```
// names of the items that are in stock
arrayMap(arrayFilter($(custom:cart), item => item.inStock), item => item.name)
```

필터링, 변환, 결합 순으로 연결하면 읽기 쉽게 구성할 수 있습니다.

```
let scores = $(custom:scores)
let passing = arrayFilter(scores, s => s >= 50)
let sorted = arraySort(passing, (a, b) => b - a)
`Top score: ${sorted[0]}`
```

실행 제한

의도하지 않은 무한 반복 등의 실수로부터 Companion을 보호하기 위해 각 표현식에는 연산 횟수(반복 횟수와 함수 호출 수) 및 중첩 함수 호출 깊이의 한도가 있습니다. 한도를 넘는 표현식은 중단되며 구문 오류와 마찬가지로 유효하지 않은 것으로 처리됩니다.

일반적인 표현식은 이 한도에 근접하지 않을 만큼 충분히 여유가 있습니다. 다만 종료 조건이 없는 while (true)나 무한 재귀 함수는 멈추지 않고 계속 실행되는 대신 실패로 처리됩니다. 입력란 표시 여부 검사처럼 표현식을 매우 자주 평가하는 곳에는 더 낮은 한도가 적용되므로 간단한 표현식을 사용하십시오.

활용 예제

항목 목록의 합계

{ price, qty } 객체 배열을 담은 사용자 변수의 합계를 계산합니다.

```
arrayReduce($(custom:cart), (sum, item) => sum + item.price * item.qty, 0)
```

범주별 개수 세기

```
let counts = { high: 0, low: 0 }
for (const score of $(custom:scores)) {
  if (score >= 50) {
    counts.high = counts.high + 1
  } else {
    counts.low = counts.low + 1
  }
}
`${counts.high} high / ${counts.low} low`
```

중첩 데이터를 안전하게 읽기

선택적 연결(?.)은 값이 없을 수 있는 경로를 안전하게 읽고, ??는 대체 값을 제공합니다.

```
$(custom:state)?.scenes?.[0]?.name ?? 'No scene'
```

요약 문자열 만들기

문장을 구분해야 한다는 점에 유의하십시오. 아래의 ;는 템플릿 문자열 줄이 윗줄의 일부로 해석되지 않게 합니다.

```
let items = $(custom:cart);
let total = arrayReduce(items, (sum, item) => sum + item.price * item.qty, 0);
`${length(items)} items, total ${total}`
```

4 원격 제어

4.1 REST API

REST API는 스크립트나 다른 소프트웨어에서 Companion을 구성할 때 사용합니다. 버전을 구분하는 정식 API이며, 앞으로 사용자 화면에서 할 수 있는 작업 대부분을 지원하도록 확장할 예정입니다. 장기적으로 기존 HTTP 원격 제어 API를 대체합니다.

참고

현재 REST API가 지원하는 범위는 제한적입니다. 이후 릴리스에서 더 많은 영역이 추가될 예정입니다.

API 켜기

REST API는 기본적으로 **꺼져 있습니다**. 다음과 같이 설정합니다.

1. **설정(Settings) > 프로토콜(Protocols)**에서 **REST API**를 켭니다.
2. 나타나는 API 키 표에서 **+**(Add API key)를 클릭합니다. 키 이름을 지정하고 권한 범위(Scopes)를 선택합니다.
3. 표시된 키를 복사합니다. 보안을 위해 키 자체를 저장하지 않으므로 다시 표시할 수 없습니다.

키 사용 방법은 인증을 참조하십시오.

기본 URL과 버전

모든 엔드포인트는 Companion 관리 화면과 같은 주소·포트의 /api/v2 아래에 있습니다.
예: http://127.0.0.1:8000/api/v2/surfaces/v1. API의 각 영역에는 개별 버전이 있습니다(/surfaces/v1의 v1).
따라서 다른 영역에 영향을 주지 않고 한 영역만 변경할 수 있습니다.

오류는 { "error": { "code": "...", "message": "..." } } 형식의 JSON으로 반환됩니다.

대화형 문서

API가 켜져 있으면 Companion에서 다음 문서를 제공합니다.

- [companion url]/api/v2/docs: 브라우저에서 요청을 직접 시험할 수 있는 대화형 문서입니다.
- [companion url]/api/v2/openapi.json: 클라이언트 코드 생성에 사용할 수 있는 OpenAPI 스키마입니다.

관련 항목: 인증, 연결 REST API, 제어 장치 REST API.

4.2 REST API 인증

REST API는 **Bearer token** 인증을 사용합니다. 모든 요청에 API 키를 포함합니다.

```
Authorization: Bearer <token>
```

curl 사용 예:

```
curl -H "Authorization: Bearer cpn_..." http://127.0.0.1:8000/api/v2/surfaces/v1
```

유효한 키가 없으면 401 Unauthorized로 거부됩니다. 키에 해당 엔드포인트가 요구하는 권한 범위가 없으면 403 Forbidden으로 거부됩니다.

API 키가 필요한 이유

Companion의 다른 기능에서는 이러한 인증을 요구하지 않을 수 있습니다. 관리 화면에는 아직 완전한 인증 기능이 없습니다. Admin UI Password는 가벼운 접근을 막는 정도입니다. 기존 HTTP 원격 제어, TCP/UDP, OSC도 네트워크로 접근할 수 있는 누구나 보낸 요청을 수락합니다.

Companion에 정식 인증을 도입하는 작업이 계획되어 있습니다. REST API가 처음부터 API 키를 요구하도록 하면, 이후 인증이 도입되어도 호환성을 깨는 변경이 필요하지 않습니다. 지금 작성한 스크립트와 연동 기능도 갑자기 새로운 인증 정보를 요구받지 않고 계속 작동할 수 있습니다.

API 키는 현재도 접근을 관리하는 데 도움이 됩니다. 스크립트나 시스템마다 별도 키를 발급하고 필요한 권한 범위만 허용할 수 있습니다. 다른 연결에 영향을 주지 않고 특정 키만 폐기할 수도 있습니다.

API 키 관리

설정(Settings) > 프로토콜(Protocols)의 **REST API** 영역에서 관리합니다. REST API를 켠 뒤에만 키 목록이 표시됩니다.

- **+**(Add API key)를 클릭해 키를 만듭니다. 나중에 식별할 수 있도록 **이름(Name)**을 지정하고 **권한 범위(Scopes)**를 선택합니다. 필요한 권한만 부여합니다.
- 생성된 키는 한 번만 표시됩니다. 즉시 복사하십시오. Companion에는 키의 해시만 저장되므로 다시 표시할 수 없습니다. 키를 잃어버리면 폐기한 뒤 새로 만듭니다.
- 표에는 각 키의 앞부분과 **마지막 사용 시각(Last used)**이 표시됩니다. 사용하지 않는 키를 찾을 때 참고할 수 있습니다.
- 연필 버튼으로 이름이나 권한 범위를 바꾸고, 휴지통 버튼으로 폐기합니다. 폐기한 키는 즉시 사용할 수 없게 됩니다.

API 키는 Admin UI Password와 별개입니다. Admin UI Password로 REST API를 보호할 수는 없습니다.

권한 범위(Scopes)

권한 범위	허용되는 작업
read	리소스 조회(GET 엔드포인트)
write	리소스 생성·수정·삭제. read 권한 포함
execute	트리거나 버튼 실행 같은 액션. read 권한 포함
connections	read 또는 write와 함께 사용하여 연결 관련 작업에 접근
admin	모든 작업. 모든 권한 범위 포함

연결 엔드포인트에는 connections와 접근 권한 범위가 모두 필요합니다. 조회에는 connections + read를, 생성·수정·삭제·이동·재시작에는 connections + write를 사용합니다. 연결의 비밀 정보도 다른 연결 구성 값과 같은 권한 규칙을 따릅니다.

4.3 연결 REST API

GET /connections/v1

모든 연결 조회

모든 연결과 현재 상태를 반환합니다. 쿼리 매개변수를 사용하면 구성 값과 비밀 정보를 응답에 포함할 수 있습니다.

필요한 권한 범위: connections, read

매개변수:

매개변수	위치	자료형	설명
include_config	쿼리	string	연결 구성 값을 응답에 포함합니다.
include_secrets	쿼리	string	연결의 비밀 정보를 응답에 포함합니다. include_config=true가 필요합니다.

예시:

GET /api/v2/connections/v1

응답 본문:


```
{
  "data": [
    {
      "id": "KJA1isEECHRDBTFjx-7tf",
      "label": "ATEM",
      "moduleId": "bmd-atem",
      "moduleVersionId": "1.2.0",
      "updatePolicy": "stable",
      "enabled": true,
      "status": {
        "category": "ok",
        "level": "info",
        "message": "Connected"
      },
      "config": {},
      "secrets": {}
    }
  ],
  "meta": {
    "total": 1,
    "limit": 100,
    "offset": 0
  }
}
```

성공 응답 (200) - 연결 목록

응답 본문(컬렉션):

필드	자료형	설명
id	string	연결 인스턴스의 고유 ID입니다.
label	string	Companion에 표시되는 연결 이름입니다.
moduleId	string	"bmd-atem", "obs-websocket"과 같은 연결 모듈 ID입니다.
moduleVersionId	string | null	이 연결에서 사용하는 모듈 버전 ID입니다. 예: "1.2.0". null이면 호환되는 최신 안정 버전을 자동으로 선택합니다. 해당 버전을 설치하는 동안에도 null이 반환될 수 있습니다.
updatePolicy	manual | stable | beta	이 연결의 모듈 버전 업데이트 정책입니다.
enabled	boolean	연결이 활성화되어 실행할 수 있는지 나타냅니다.
status	object | null	연결에서 보고한 최신 실행 상태입니다.
↳ category	string | null	연결 모듈에서 보고한 상태 분류입니다.
↳ level	string | null	현재 연결 상태의 심각도입니다.
↳ message	string | null	사용자가 읽을 수 있는 연결 모듈의 상태 메시지입니다.
config	object	비밀 정보가 아닌 연결 구성 값입니다.
secrets	object	연결 구성 중 비밀 정보에 해당하는 값입니다.

POST /connections/v1

연결 생성

지정한 연결 모듈 ID로 새 연결 인스턴스를 만듭니다. 모듈이 설치되어 있지 않으면, 기본적으로 호환되는 최신 안정 버전의 설치를 대기열에 등록합니다.

필요한 권한 범위: connections, write

요청 본문 (application/json):

필드	자료형	필수	설명
moduleId	string	예	생성할 연결의 모듈 ID입니다. 예: "bmd-atem", "obs-websocket".
label	string	예	새 연결의 표시 이름입니다.

필드	자료형	필수	설명
versionId	string | null	아니요	사용할 모듈 버전을 지정합니다. 생략하거나 null로 설정하면 호환되는 최신 안정 버전을 사용합니다. 모듈이 설치되어 있지 않으면 해당 버전의 설치를 대기열에 등록합니다.
updatePolicy	manual | stable | beta	아니요	새 연결의 모듈 버전 업데이트 정책입니다.
disabled	boolean	아니요	새 연결을 비활성화된 상태로 만들지 지정합니다.

예시:

POST /api/v2/connections/v1

요청 본문:

```
{
  "moduleId": "bmd-atem",
  "label": "ATEM",
  "versionId": null,
  "updatePolicy": "stable",
  "disabled": false
}
```

응답 본문:

```
{
  "data": {
    "id": "KJA1isEECHRDBTFjx-7tf"
  }
}
```

성공 응답 - 연결 생성 완료

응답 본문:

필드	자료형	설명
id	string	연결 인스턴스의 고유 ID입니다.

GET /connections/v1/tree

연결 배치 조회

전체 컬렉션 계층에 따라 그룹화된 모든 연결을 반환합니다. 배열 순서는 현재 화면의 표시 순서와 같습니다.

필요한 권한 범위: connections, read

매개변수:

매개변수	위치	자료형	설명
include_config	쿼리	string	연결 구성 값을 응답에 포함합니다.
include_secrets	쿼리	string	연결의 비밀 정보를 응답에 포함합니다. include_config=true가 필요합니다.

예시:

GET /api/v2/connections/v1/tree

응답 본문:


```
{
  "data": {
    "connections": [
      {
        "id": "KJA1isEECHRDBTFjx-7tf",
        "label": "ATEM",
        "moduleId": "bmd-atem",
        "moduleVersionId": "1.2.0",
        "updatePolicy": "stable",
        "enabled": true,
        "status": {
          "category": "ok",
          "level": "info",
          "message": "Connected"
        },
        "config": {},
        "secrets": {}
      }
    ],
    "collections": [null]
  }
}
```


성공 응답 (200) - 연결 배치

응답 본문:

필드	자료형	설명
connections	object[]	그룹에 속하지 않은 최상위 연결을 현재 표시 순서대로 나열합니다.
collections	object[]	최상위 컬렉션을 현재 표시 순서대로 나열합니다.

PATCH /connections/v1/move

연결 이동

여러 연결의 이동을 하나의 작업으로 처리합니다. 앞선 작업의 결과를 기준으로 요청 순서대로 각 작업을 평가하며, 전체 변경을 한 번에 반영합니다.

필요한 권한 범위: connections, write

요청 본문 (application/json):

필드	자료형	필수	설명
moves	object[]	예	순서대로 평가한 뒤 전체 변경을 한 번에 저장할 이동 작업 목록입니다.

예시:

PATCH /api/v2/connections/v1/move

요청 본문:

```
{
  "moves": [
    {
      "connectionId": "string",
      "collectionId": null,
      "position": 1
    }
  ]
}
```

성공 응답 - 연결 이동 완료

GET /connections/v1/{connectionId}

연결 하나 조회

ID로 연결 하나를 조회하여 구성 값과 현재 상태를 반환합니다.

필요한 권한 범위: connections, read

매개변수:

매개변수	위치	자료형	설명
connectionId	경로	string	연결 목록 조회 또는 연결 생성 엔드포인트에서 반환한 연결 인스턴스 ID입니다.
include_secrets	쿼리	string	연결의 비밀 정보를 응답에 포함합니다.

예시:

GET /api/v2/connections/v1/KJA1isEECHRDBTFjx-7tf

응답 본문:


```
{
  "data": {
    "id": "KJA1isEECHRDBTFjx-7tf",
    "label": "ATEM",
    "moduleId": "bmd-atem",
    "moduleVersionId": "1.2.0",
    "updatePolicy": "stable",
    "enabled": true,
    "status": {
      "category": "ok",
      "level": "info",
      "message": "Connected"
    },
    "config": {},
    "secrets": {}
  }
}
```


성공 응답 (200) - 연결 상세 정보

응답 본문:

필드	자료형	설명
id	string	연결 인스턴스의 고유 ID입니다.
label	string	Companion에 표시되는 연결 이름입니다.
moduleId	string	"bmd-atem", "obs-websocket"과 같은 연결 모듈 ID입니다.
moduleVersionId	string | null	이 연결에서 사용하는 모듈 버전 ID입니다. 예: "1.2.0". null이면 호환되는 최신 안정 버전을 자동으로 선택합니다. 해당 버전을 설치하는 동안에도 null이 반환될 수 있습니다.
updatePolicy	manual | stable | beta	이 연결의 모듈 버전 업데이트 정책입니다.
enabled	boolean	연결이 활성화되어 실행할 수 있는지 나타냅니다.
status	object | null	연결에서 보고한 최신 실행 상태입니다.
↳ category	string | null	연결 모듈에서 보고한 상태 분류입니다.
↳ level	string | null	현재 연결 상태의 심각도입니다.
↳ message	string | null	사용자가 읽을 수 있는 연결 모듈의 상태 메시지입니다.
config	object	비밀 정보가 아닌 연결 구성 값입니다.
secrets	object	연결 구성 중 비밀 정보에 해당하는 값입니다.

PATCH /connections/v1/{connectionId}

연결 수정

연결 구성의 일부를 수정합니다. 변경할 필드만 전송합니다.

필요한 권한 범위: connections, write

매개변수:

매개변수	위치	자료형	설명
connectionId	경로	string	연결 목록 조회 또는 연결 생성 엔드포인트에서 반환한 연결 인스턴스 ID입니다.

요청 본문 (application/json):

필드	자료형	필수	설명
label	string	아니요	연결에 지정할 새 표시 이름입니다.
disabled	boolean	아니요	true이면 연결을 비활성화하고, false이면 활성화합니다.
config	object	아니요	현재 연결 구성에 병합할 일반 구성 값입니다.
secrets	object	아니요	현재 연결의 비밀 정보에 병합할 구성 값입니다.
updatePolicy	manual | stable | beta	아니요	적용할 모듈 버전 업데이트 정책입니다.
versionId	string | null	아니요	사용할 모듈 버전입니다. 예: "1.2.0". 생략하면 현재 버전을 유지합니다. null로 설정하면 호환되는 최신 안정 버전으로 전환합니다.

예시:

PATCH /api/v2/connections/v1/KJA1isEECHRDBTFjx-7tf

요청 본문:

```
{
  "label": "ATEM Program",
  "disabled": false,
  "config": {
    "host": "10.50.0.20",
    "fadeFps": 10
  },
  "secrets": {},
  "updatePolicy": "manual",
  "versionId": "1.2.0"
}
```

응답 본문:


```
{
  "data": {
    "id": "KJA1isEECHRDBTFjx-7tf",
    "label": "ATEM",
    "moduleId": "bmd-atem",
    "moduleVersionId": "1.2.0",
    "updatePolicy": "stable",
    "enabled": true,
    "status": {
      "category": "ok",
      "level": "info",
      "message": "Connected"
    },
    "config": {},
    "secrets": {}
  }
}
```


성공 응답 (200) - 수정된 연결

응답 본문:

필드	자료형	설명
id	string	연결 인스턴스의 고유 ID입니다.
label	string	Companion에 표시되는 연결 이름입니다.
moduleId	string	"bmd-atem", "obs-websocket"과 같은 연결 모듈 ID입니다.
moduleVersionId	string | null	이 연결에서 사용하는 모듈 버전 ID입니다. 예: "1.2.0". null이면 호환되는 최신 안정 버전을 자동으로 선택합니다. 해당 버전을 설치하는 동안에도 null이 반환될 수 있습니다.
updatePolicy	manual | stable | beta	이 연결의 모듈 버전 업데이트 정책입니다.
enabled	boolean	연결이 활성화되어 실행할 수 있는지 나타냅니다.
status	object | null	연결에서 보고한 최신 실행 상태입니다.
↳ category	string | null	연결 모듈에서 보고한 상태 분류입니다.
↳ level	string | null	현재 연결 상태의 심각도입니다.
↳ message	string | null	사용자가 읽을 수 있는 연결 모듈의 상태 메시지입니다.
config	object	비밀 정보가 아닌 연결 구성 값입니다.
secrets	object	연결 구성 중 비밀 정보에 해당하는 값입니다.

DELETE /connections/v1/{connectionId}

연결 삭제

연결과 그 연결에 속한 모든 구성을 삭제합니다.

필요한 권한 범위: connections, write

매개변수:

매개변수	위치	자료형	설명
connectionId	경로	string	연결 목록 조회 또는 연결 생성 엔드포인트에서 반환한 연결 인스턴스 ID입니다.

예시:

DELETE /api/v2/connections/v1/KJA1isEECHRDBTFjx-7tf

성공 응답 - 연결 삭제 완료

GET /connections/v1/{connectionId}/config-fields

연결 구성 필드 정의 조회

연결 모듈의 구성 필드 정의를 반환합니다. 필드 자료형, 제약 조건, 선택 가능한 옵션이 포함됩니다.
연결이 실행 중이어야 합니다.

필요한 권한 범위: connections, read

매개변수:

매개변수	위치	자료형	설명
connectionId	경로	string	연결 목록 조회 또는 연결 생성 엔드포인트에서 반환한 연결 인스턴스 ID입니다.

예시:

GET /api/v2/connections/v1/KJA1isEECHRDBTFjx-7tf/config-fields

응답 본문:

```
{
  "data": [null]
}
```

성공 응답 (200) - 구성 필드 정의

POST /connections/v1/{connectionId}/restart

연결 다시 시작

연결 프로세스를 강제로 다시 시작합니다. 연결이 비활성화되어 있으면 실패합니다.

필요한 권한 범위: connections, write

매개변수:

매개변수	위치	자료형	설명
connectionId	경로	string	연결 목록 조회 또는 연결 생성 엔드포인트에서 반환한 연결 인스턴스 ID입니다.

예시:

POST /api/v2/connections/v1/KJA1isEECHRDBTFjx-7tf/restart

응답 본문:

```
{
  "data": {
    "id": "KJA1isEECHRDBTFjx-7tf",
    "message": "Restart triggered"
  }
}
```

성공 응답 (200) - 재시작 요청 실행

응답 본문:

필드	자료형	설명
id	string	다시 시작한 연결 인스턴스 ID입니다.
message	string	재시작 요청을 확인하는 메시지입니다.

4.4 제어 장치 REST API

GET /surfaces/v1

모든 제어 장치 조회

현재 연결 여부와 관계없이 등록된 모든 제어 장치와 현재 상태를 반환합니다.

필요한 권한 범위: read

예시:

GET /api/v2/surfaces/v1

응답 본문:


```
{
  "data": [
    {
      "id": "streamdeck:1A2B3C4D",
      "type": "Elgato Stream Deck XL",
      "integrationType": "elgato-stream-deck",
      "name": "Front of house",
      "displayName": "Front of house (streamdeck:1A2B3C4D)",
      "isConnected": true,
      "size": {
        "rows": 4,
        "columns": 8
      },
      "brightness": 80,
      "page": {
        "id": "ggmHXCUQ0RRXUwEr8HHtQ",
        "number": 1,
        "name": "Main"
      },
      "groupId": "streamdeck:1A2B3C4D"
    }
  ],
  "meta": {
    "total": 1,
    "limit": 100,
    "offset": 0
  }
}
```


성공 응답 (200) - 제어 장치 목록

응답 본문(컬렉션):

필드	자료형	설명
id	string	제어 장치의 고유 ID입니다.
type	string	연동 기능에서 보고한 제어 장치 모델입니다.
integrationType	string	제어 장치가 사용하는 연동 기능입니다.
name	string	Companion에서 제어 장치에 지정한 이름입니다.
displayName	string	Companion 화면에 표시되는 제어 장치 이름입니다.
isConnected	boolean	제어 장치가 현재 연결되어 있는지 나타냅니다.
size	object | null	알려진 경우 제어 장치의 버튼 격자 크기입니다.
↳ rows	number	제어 장치의 버튼 행 수입니다.
↳ columns	number	제어 장치의 버튼 열 수입니다.

필드	자료형	설명
brightness	number | null	제어 장치의 밝기를 백분율로 나타냅니다. 제어 장치 구성에 밝기가 지정되어 있지 않으면 null입니다.
page	object | null	제어 장치에 현재 표시되는 페이지입니다. 표시 중인 페이지가 없으면 null입니다.
↳ id	string	페이지의 고유 ID입니다.
↳ number	number | null	페이지 목록에서 해당 페이지의 위치입니다. 목록에서 제거된 페이지이면 null입니다.
↳ name	string | null	페이지의 표시 이름입니다.
groupId	string	제어 장치가 속한 그룹의 ID입니다. 그룹에 속하지 않으면 일반적으로 장치의 id와 같습니다.

GET /surfaces/v1/{surfaceId}

제어 장치 하나 조회

현재 연결 여부와 관계없이 ID로 제어 장치 하나를 조회하여 현재 상태를 반환합니다.

필요한 권한 범위: read

매개변수:

매개변수	위치	자료형	설명
surfaceId	경로	string	제어 장치 목록 조회 엔드포인트에서 반환한 제어 장치 ID입니다.

예시:

GET /api/v2/surfaces/v1/streamdeck:1A2B3C4D

응답 본문:


```
{
  "data": {
    "id": "streamdeck:1A2B3C4D",
    "type": "Elgato Stream Deck XL",
    "integrationType": "elgato-stream-deck",
    "name": "Front of house",
    "displayName": "Front of house (streamdeck:1A2B3C4D)",
    "isConnected": true,
    "size": {
      "rows": 4,
      "columns": 8
    },
    "brightness": 80,
    "page": {
```

```
      "id": "ggmHXCUQ0RRXUwEr8HHtQ",
      "number": 1,
      "name": "Main"
    },
    "groupId": "streamdeck:1A2B3C4D"
  }
}
```


성공 응답 (200) - 요청한 제어 장치

응답 본문:

필드	자료형	설명
id	string	제어 장치의 고유 ID입니다.
type	string	연동 기능에서 보고한 제어 장치 모델입니다.
integrationType	string	제어 장치가 사용하는 연동 기능입니다.
name	string	Companion에서 제어 장치에 지정한 이름입니다.
displayName	string	Companion 화면에 표시되는 제어 장치 이름입니다.
isConnected	boolean	제어 장치가 현재 연결되어 있는지 나타냅니다.
size	object | null	알려진 경우 제어 장치의 버튼 격자 크기입니다.
↳ rows	number	제어 장치의 버튼 행 수입니다.
↳ columns	number	제어 장치의 버튼 열 수입니다.
brightness	number | null	제어 장치의 밝기를 백분율로 나타냅니다. 제어 장치 구성에 밝기가 지정되어 있지 않으면 null입니다.
page	object | null	제어 장치에 현재 표시되는 페이지입니다. 표시 중인 페이지가 없으면 null입니다.
↳ id	string	페이지의 고유 ID입니다.
↳ number	number | null	페이지 목록에서 해당 페이지의 위치입니다. 목록에서 제거된 페이지이면 null입니다.
↳ name	string | null	페이지의 표시 이름입니다.
groupId	string	제어 장치가 속한 그룹의 ID입니다. 그룹에 속하지 않으면 일반적으로 장치의 id와 같습니다.

PATCH /surfaces/v1/{surfaceId}

제어 장치 수정

연결된 제어 장치를 수정합니다. 현재는 밝기만 변경할 수 있습니다.

필요한 권한 범위: write

매개변수:

매개변수	위치	자료형	설명
surfaceId	경로	string	제어 장치 목록 조회 엔드포인트에서 반환한 제어 장치 ID입니다.

요청 본문 (application/json):

필드	자료형	필수	설명
brightness	integer	예	제어 장치에 적용할 밝기를 백분율로 지정합니다.

예시:

PATCH /api/v2/surfaces/v1/streamdeck:1A2B3C4D

요청 본문:

```
{
  "brightness": 50
}
```

응답 본문:

```
{
  "data": {
    "id": "streamdeck:1A2B3C4D",
    "type": "Elgato Stream Deck XL",
    "integrationType": "elgato-stream-deck",
    "name": "Front of house",
    "displayName": "Front of house (streamdeck:1A2B3C4D)",
    "isConnected": true,
    "size": {
      "rows": 4,
      "columns": 8
    },
    "brightness": 80,
    "page": {
      "id": "ggmHXCUQ0RRXUwEr8HHtQ",
      "number": 1,
      "name": "Main"
    },
    "groupId": "streamdeck:1A2B3C4D"
  }
}
```

성공 응답 (200) - 수정된 제어 장치

응답 본문:

필드	자료형	설명
id	string	제어 장치의 고유 ID입니다.
type	string	연동 기능에서 보고한 제어 장치 모델입니다.
integrationType	string	제어 장치가 사용하는 연동 기능입니다.
name	string	Companion에서 제어 장치에 지정한 이름입니다.
displayName	string	Companion 화면에 표시되는 제어 장치 이름입니다.
isConnected	boolean	제어 장치가 현재 연결되어 있는지 나타냅니다.
size	object | null	알려진 경우 제어 장치의 버튼 격자 크기입니다.
↳ rows	number	제어 장치의 버튼 행 수입니다.
↳ columns	number	제어 장치의 버튼 열 수입니다.
brightness	number | null	제어 장치의 밝기를 백분율로 나타냅니다. 제어 장치 구성에 밝기가 지정되어 있지 않으면 null입니다.
page	object | null	제어 장치에 현재 표시되는 페이지입니다. 표시 중인 페이지가 없으면 null입니다.
↳ id	string	페이지의 고유 ID입니다.
↳ number	number | null	페이지 목록에서 해당 페이지의 위치입니다. 목록에서 제거된 페이지이면 null입니다.
↳ name	string | null	페이지의 표시 이름입니다.
groupId	string	제어 장치가 속한 그룹의 ID입니다. 그룹에 속하지 않으면 일반적으로 장치의 id와 같습니다.

4.5 TCP / UDP 제어

TCP 또는 UDP 명령을 보내 원격으로 실행할 수 있습니다. 먼저 설정에서 사용할 포트 번호를 지정해야 합니다.

주의

TCP 명령 끝에는 줄바꿈 문자, 즉 \n (0x0A) 또는 \r\n (0x0D, 0x0A)을 붙여야 합니다.

명령

- SURFACE <surface id> PAGE-SET <page number>
  제어장치를 지정 페이지로 전환
- SURFACE <surface id> PAGE-UP
  지정 제어장치의 페이지 번호 증가
- SURFACE <surface id> PAGE-DOWN
  지정 제어장치의 페이지 번호 감소
- LOCATION <page>/<row>/<column> PRESS
  버튼을 눌렀다가 놓기(누름·놓기 액션 모두 실행)
- LOCATION <page>/<row>/<column> DOWN
  버튼 누르기(누름 액션 실행)
- LOCATION <page>/<row>/<column> UP
  버튼 놓기(놓기 액션 실행)
- LOCATION <page>/<row>/<column> ROTATE-LEFT
  버튼/인코더의 왼쪽 회전 실행
- LOCATION <page>/<row>/<column> ROTATE-RIGHT
  버튼/인코더의 오른쪽 회전 실행
- LOCATION <page>/<row>/<column> SET-STEP <step>
  버튼/인코더의 현재 단계 설정
- LOCATION <page>/<row>/<column> STYLE TEXT <text>
  버튼 텍스트 변경
- LOCATION <page>/<row>/<column> STYLE COLOR <color HEX>
  버튼 글자색 변경(#000000)
- LOCATION <page>/<row>/<column> STYLE BGCOLOR <color HEX>
  버튼 배경색 변경(#000000)
- CUSTOM-VARIABLE <name> SET-VALUE <value>
  사용자 변수 값 변경
- CUSTOM-VARIABLE <name> GET-VALUE 사용자 변수 값 조회
- SURFACES RESCAN USB 제어장치를 다시 검색

예제

에뮬레이터 제어장치를 23페이지로 전환:
SURFACE emulator PAGE-SET 23

1페이지 2행 3열 버튼 누르기:
LOCATION 1/2/3 PRESS

사용자 변수 "cue"의 값을 "intro"로 변경:
CUSTOM-VARIABLE cue SET-VALUE intro

사용 중단 예정 명령

주의

이 명령은 향후 Companion 버전에서 제거됩니다. 위 목록에 같은 기능을 제공하는 명령이 있습니다.

다음 명령은 사용 중단 예정이며, 위 목록의 명령으로 대체할 수 있습니다. 사용하려면 설정에서 지원 기능을 켜야 합니다.

- PAGE-SET <page number> <surface id>
  장치를 지정 페이지로 전환
- PAGE-UP <surface id>
  지정 장치의 페이지 번호 증가
- PAGE-DOWN <surface id>
  지정 제어장치의 페이지 번호 감소
- BANK-PRESS <page> <bank>
  버튼을 눌렀다가 놓기(누름·놓기 액션 모두 실행)
- BANK-DOWN <page> <bank>
  버튼 누르기(누름 액션 실행)
- BANK-UP <page> <bank>
  버튼 놓기(놓기 액션 실행)
- STYLE BANK <page> <bank> TEXT <text>
  버튼 텍스트 변경
- STYLE BANK <page> <bank> COLOR <color HEX>
  버튼 글자색 변경(#000000)
- STYLE BANK <page> <bank> BGCOLOR <color HEX>
  버튼 배경색 변경(#000000)
- RESCAN 새로 연결한 USB 제어장치를 다시 검색

4.6 HTTP 원격 제어

Companion이 실행 중인 IP 주소와 포트로 HTTP 요청을 보내 원격으로 실행할 수 있습니다.

도움말

Companion 5.1에는 스크립트나 다른 소프트웨어에서 Companion을 설정하는 새로운 REST API가 도입되었습니다.
앞으로 기능을 확장하여 이 API를 대체할 예정입니다. 현재는 지원하는 기능이 서로 다르므로 두 API를 함께 사용할 수 있습니다.

명령

이 API는 REST 원칙을 따르며, 일반적으로 POST 요청으로 값을 변경하고 GET 요청으로 값을 조회합니다.

- 버튼을 눌렀다가 놓기(누름·놓기 액션 모두 실행)
  메서드: POST
  경로: /api/location/<page>/<row>/<column>/press

- 버튼을 누른 상태로 유지(누름 액션 실행)
  메서드: POST
  경로: /api/location/<page>/<row>/<column>/down
- 버튼 놓기(놓기 액션 실행)
  메서드: POST
  경로: /api/location/<page>/<row>/<column>/up
- 버튼/인코더의 왼쪽 회전 실행 메서드: POST
  경로: /api/location/<page>/<row>/<column>/rotate-left
- 버튼/인코더의 오른쪽 회전 실행 메서드: POST
  경로: /api/location/<page>/<row>/<column>/rotate-right
- 버튼/인코더의 현재 단계 설정 메서드: POST
  경로: /api/location/<page>/<row>/<column>/step
- 버튼 배경색 변경 메서드: POST
  경로: /api/location/<page>/<row>/<column>/style?bgcolor=<bgcolor HEX>
- 버튼 배경색 변경 메서드: POST
  경로: /api/location/<page>/<row>/<column>/style
  본문: { "bgcolor": "<bgcolor HEX>" } 또는 { "bgcolor": "rgb(<red>,<green>,<blue>)" }
- 버튼 글자색 변경 메서드: POST
  경로: /api/location/<page>/<row>/<column>/style?color=<color HEX>
- 버튼 글자색 변경 메서드: POST
  경로: /api/location/<page>/<row>/<column>/style
  본문: { "color": "<color HEX>" } 또는 { "color": "rgb(<red>,<green>,<blue>)" }
- 버튼 텍스트 변경 메서드: POST
  경로: /api/location/<page>/<row>/<column>/style?text=<text>
- 버튼 텍스트 변경 메서드: POST
  경로: /api/location/<page>/<row>/<column>/style
  본문: { "text": "<text>" }
- 사용자 변수 값 변경
  메서드: POST
  경로: /api/custom-variable/<name>/value?value=<value>
- 사용자 변수 값 변경
  메서드: POST
  경로: /api/custom-variable/<name>/value
  본문: <value>
- 사용자 변수 값 조회
  메서드: GET
  경로: /api/custom-variable/<name>/value

- 모듈 변수 값 조회
  메서드: GET
  경로: /api/variable/<Connection Label>/<name>/value
- USB 제어장치 다시 검색 메서드: POST 경로: /api/surfaces/rescan

연결 관리

- 모든 연결과 현재 상태 조회 메서드: GET 경로: /api/connections 응답: 연결 객체의 JSON 배열

```
[
 {
  "id": "abc123",
  "label": "My OBS",
  "moduleId": "obs-websocket",
  "enabled": true,
  "sortOrder": 0,
  "status": { "category": "good", "level": "ok", "message": "Connected" }
 }
]
```

- 특정 연결의 상태 조회 메서드: GET 경로: /api/connections/<id>/status 응답(200):

```
{
 "id": "abc123",
 "label": "My OBS",
 "enabled": true,
 "status": { "category": "good", "level": "ok", "message": "Connected" }
}
```

  오류(404): { "status": 404, "message": "Connection not found" }
- 연결 프로세스 재시작 메서드: POST 경로: /api/connections/<id>/restart 응답(200): { "id": "abc123", "message": "Restart triggered" } 오류(404): { "status": 404, "message": "Connection not found" } 오류(409): { "status": 409, "message": "Connection is inactive and cannot be restarted" }
- 연결 활성화 메서드: POST 경로: /api/connections/<id>/enable 응답(200): { "id": "abc123", "enabled": true } 오류(404): { "status": 404, "message": "Connection not found" }
- 연결 비활성화 메서드: POST 경로: /api/connections/<id>/disable 응답(200): { "id": "abc123", "enabled": false } 오류(404): { "status": 404, "message": "Connection not found" }

예제

1페이지 0행 2열 버튼 누르기:
POST /api/location/1/0/2/press

1페이지의 첫 번째 행, 다섯 번째 열(행 인덱스 0, 열 인덱스 4)의 텍스트를 TEST로 변경:
POST /api/location/1/0/4/style?text=TEST

2페이지 1행 4열의 텍스트를 TEST, 배경색을 #ffffff, 글자색을 #000000, 글자 크기를 28px로 변경:
POST /api/location/2/1/4/style
본문: { "text": "TEST", "bgcolor": "#ffffff", "color": "#000000", "size": 28 }

URL로 사용자 변수 "cue"의 값을 "intro"로 변경:
POST /api/custom-variable/cue/value?value=intro

요청 본문으로 사용자 변수 "cue"의 값을 "Some text"로 변경:
POST /api/custom-variable/cue/value
Content-Type text/plain
본문: Some text

JSON 요청 본문으로 사용자 변수 "cue"의 값을 "Some text"로 변경:
POST /api/custom-variable/cue/value
Content-Type application/json
본문: "Some text" - 텍스트를 큰따옴표로 감싸야 하며, 텍스트 안의 따옴표는 백슬래시로 이스케이프해야 합니다.

사용자 변수 "data"를 JSON 객체로 변경: POST /api/custom-variable/data/value
Content-Type application/json
본문: {"name":"Douglas", "answer":42} - 요청 본문은 유효한 JSON이어야 합니다.
객체는 문자열로 변환되지 않고 변수 값에 그대로 저장됩니다. boolean, number, array, null 형식도 사용할 수 있습니다.
JSON은 undefined 값을 전송할 수 없지만, Companion은 빈 요청 본문을 undefined로 해석합니다.
객체의 속성 값은 undefined일 수 있습니다.

모든 연결 조회: GET /api/connections

특정 연결의 상태 조회: GET /api/connections/abc123/status

연결 재시작: POST /api/connections/abc123/restart

비활성화된 연결 켜기: POST /api/connections/abc123/enable

연결 끄기: POST /api/connections/abc123/disable

사용 중단 예정 명령

주의

이 명령은 향후 Companion 버전에서 제거됩니다. 위 목록에 같은 기능을 제공하는 명령이 있습니다.

다음 명령은 사용 중단 예정이며, 위 목록의 명령으로 대체할 수 있습니다. 사용하려면 설정에서 지원 기능을 켜야 합니다.

- /press/bank/<page>/<bank>
  버튼을 눌렀다가 놓기(누름·놓기 액션 모두 실행)
- /press/bank/<page>/<bank>/down
  버튼 누르기(누름 액션 모두 실행)
- /press/bank/<page>/<bank>/up
  버튼 놓기(놓기 액션 실행)
- /style/bank/<page>/<bank>?bgcolor=<bgcolor HEX>
  버튼 배경색 변경
- /style/bank/<page>/<bank>?color=<color HEX>
  버튼 글자색 변경

- /style/bank/<page>/<bank>?text=<text>
  버튼 텍스트 변경
- /style/bank/<page>/<bank>?size=<text size>
  버튼 글자 크기 변경(사전 정의된 값 중 선택)
- /set/custom-variable/<name>?value=<value>
  사용자 변수 값 변경
- /rescan
  새로 연결한 USB 제어장치를 다시 검색

4.7 OSC 제어

포트 12321로 OSC 명령을 보내 원격으로 실행할 수 있습니다. 포트 번호는 변경할 수 있습니다.

명령

- /location/<page>/<row>/<column>/press
  버튼을 눌렀다가 놓기(누름·놓기 액션 모두 실행)
- /location/<page>/<row>/<column>/down
  버튼을 누른 상태로 유지(누름 액션 실행)
- /location/<page>/<row>/<column>/up
  버튼 놓기(놓기 액션 실행)
- /location/<page>/<row>/<column>/rotate-left
  버튼/인코더의 왼쪽 회전 실행
- /location/<page>/<row>/<column>/rotate-right
  버튼/인코더의 오른쪽 회전 실행
- /location/<page>/<row>/<column>/step
  버튼/인코더의 현재 단계 설정
- /location/<page>/<row>/<column>/style/bgcolor <red 0-255> <green 0-255> <blue 0-255>
  버튼 배경색 변경
- /location/<page>/<row>/<column>/style/bgcolor <css color>
  버튼 배경색 변경
- /location/<page>/<row>/<column>/style/color <red 0-255> <green 0-255> <blue 0-255>
  버튼 글자색 변경
- /location/<page>/<row>/<column>/style/color <css color>
  버튼 글자색 변경
- /location/<page>/<row>/<column>/style/text <text>
  버튼 텍스트 변경
- /custom-variable/<name>/value <value>
  사용자 변수 값 변경

- /surfaces/rescan USB 제어장치 다시 검색

예제

1페이지 0행 5열 버튼을 누른 뒤 즉시 놓기
/location/1/0/5/press

1페이지 0행 5열 버튼의 배경색을 빨간색으로 변경
/location/1/0/5/style/bgcolor 255 0 0
/location/1/0/5/style/bgcolor rgb(255,0,0)
/location/1/0/5/style/bgcolor #ff0000

1페이지 0행 5열의 텍스트를 ONLINE으로 변경
/location/1/0/5/style/text ONLINE

사용자 변수 "cue"의 값을 "intro"로 변경:
/custom-variable/cue/value intro

사용 중단 예정 명령

주의

이 명령은 향후 Companion 버전에서 제거됩니다. 위 목록에 같은 기능을 제공하는 명령이 있습니다.

다음 명령은 사용 중단 예정이며, 위 목록의 명령으로 대체할 수 있습니다. 사용하려면 설정에서 지원 기능을 켜야 합니다.

- /press/bank/ <page> <bank>
  버튼을 눌렀다가 놓기(누름·놓기 액션 모두 실행)
- /press/bank/ <page> <bank> <1>
  버튼을 누른 상태로 유지(누름 액션 실행)
- /press/bank/ <page> <bank> <0>
  버튼 놓기(놓기 액션 실행)
- /style/bgcolor/ <page> <bank> <red 0-255> <green 0-255> <blue 0-255>
  버튼 배경색 변경
- /style/color/ <page> <bank> <red 0-255> <green 0-255> <blue 0-255>
  버튼 글자색 변경
- /style/text/ <page> <bank> <text>
  버튼 텍스트 변경
- /rescan 1 새로 연결한 USB 제어장치를 다시 검색

4.8 Prometheus로 모니터링

Companion은 서버와 애플리케이션 모니터링에 널리 쓰이는 Prometheus에 지표를 제공할 수 있습니다. 이를 통해 다른 시스템과 함께 Companion의 상태를 확인하고, 높은 메모리 사용량이나 느린 화면 그리기 등의 문제를 조사할 수 있습니다.

참고

현재 제공하는 지표는 Companion 개발자들이 테스트 환경을 모니터링하며 유용하다고 판단한 항목입니다.
실제 운영 환경에서 얼마나 유용한지는 아직 확인 중입니다. 추가로 필요한 지표가 있다면 알려 주십시오. 반영을 검토하겠습니다.

지표 엔드포인트 활성화

지표 엔드포인트는 기본으로 **비활성화**되어 있습니다.

1. **설정(Settings) > 프로토콜(Protocols)**에서 **Prometheus metrics endpoint**를 켭니다.
2. **수집 토큰(Scrape token)**이 생성됩니다. Prometheus가 지표를 읽으려면 이 토큰을 전송해야 합니다.

토큰 아래에 Prometheus 설정 예제가 표시됩니다. 자신의 Prometheus 설정으로 복사한 뒤 환경에 맞게 수정할 수 있습니다.
예제는 다음과 같습니다.

```
scrape_configs:
  - job_name: companion
    metrics_path: /api/metrics
    authorization:
      credentials: <your scrape token>
    static_configs:
      - targets: ['<companion address>:8000']
```

지표는 Companion 관리 화면과 같은 주소·포트의 /api/metrics에서 제공됩니다. 올바른 토큰이 없는 요청은 거부됩니다.
토큰을 변경하려면 옆의 재생성 버튼을 사용하십시오. 기존 토큰은 즉시 무효화됩니다.

수집 토큰은 REST API 키와 별개입니다.

측정 항목

Companion 프로세스의 메모리·CPU 사용량과 처리 부하 등 기본 지표 외에도, 다음과 같이 이름이 companion_으로 시작하는 지표를 제공합니다.

- Companion 버전(companion_build_info)
- 버튼, 페이지, 트리거, 변수, 연결, 제어장치 수
- 연결별 실행 여부, 재시작 빈도, 변수·피드백 수, Companion과의 통신량
- 버튼을 그리는 작업량과 소요 시간
- Companion 데이터베이스의 크기와 처리 부하

Grafana 대시보드

이 지표를 표시하는 Grafana 대시보드와 가져오기 방법을 Companion 저장소에서 제공합니다.

4.9 Artnet / DMX 제어

Companion은 Artnet 메시지를 수신하여 조명 콘솔 등 Artnet 소스에서 버튼을 누르고 액션을 실행할 수 있게 합니다.

활성화

Artnet 제어를 사용하려면 먼저 설정(Settings)에서 기능을 켜야 합니다.

관련 설정은 다음과 같습니다.

- **Artnet Listener**: Artnet으로 Companion을 제어할 수 있도록 켭니다.
- **Artnet Universe**: Companion이 수신할 Artnet 유니버스입니다. 번호는 0부터 시작하므로 첫 유니버스는 0입니다.
- **Artnet Channel**: 해당 유니버스에서 Companion 버튼에 대응시킬 첫 DMX 채널입니다.

채널 대응 관계

Companion의 각 버튼은 DMX 채널에 대응합니다. 채널 값에 따라 버튼 누름 여부가 결정됩니다.

- 값이 0이면 버튼을 누르지 않은 상태입니다.
- 0이 아닌 값이면 버튼을 실행합니다. 버튼을 눌렀다가 놓는 동작과 같습니다.

설정한 **Artnet Channel**부터 버튼 격자를 행 단위로 가로 방향으로 따라가며 채널이 순서대로 대응합니다.

조명 기기 정의 파일

Companion의 Artnet 설정 탭 아래쪽에 GrandMA2용 기기 정의 파일(fixture file) 예제가 있습니다.
조명 콘솔로 가져오면 채널에 올바른 이름을 표시할 수 있습니다.

4.10 EmberPlus 제어

EmberPlus에서 다음 항목을 제어할 수 있습니다.

Companion 제품 정보	
경로:	/Companion Tree/identity/<parameter name>
권한:	Read Only
매개변수:	product, company, version, build
매개변수 형식:	string

버튼 조작	
경로:	/Companion Tree/pages/<page name>/<button number>/<parameter name>
경로:	/Companion Tree/location/<page number>/<row number>/<column number>/<parameter name>
권한:	Read/Write
매개변수:	State, Label, Text_Color, Background_Color
매개변수 형식:	boolean, string

내부 변수	
경로:	/Companion Tree/variables/internal/<parameter name>/<parameter type>

내부 변수	
권한:	Read Only
매개변수 형식:	boolean, integer, string

사용자 변수	
경로:	/Companion Tree/variables/custom/<parameter name>/string
권한:	Read/Write
매개변수 형식:	string

액션 기록기	
경로:	/Companion Tree/action recorder/<parameter name>
권한:	Read/Write
매개변수:	Enable, Discard
매개변수 형식:	boolean

제공자 재시작

다음 상황에서는 Ember Plus 제공자가 자동으로 재시작하여 Ember 트리를 다시 구성합니다.

- 페이지 수 변경
- 버튼 격자 크기 변경
- 새 연결 추가
- 연결의 이름 변경
- 새 사용자 변수 추가

이때 클라이언트 연결이 끊기므로 실제 운영 중에는 이러한 변경을 피해야 합니다.

노드 경로의 안정성

변수의 숫자 경로는 초기화 순서에 따라 달라지므로, Ember Plus 서버는 Companion을 재시작한 뒤에도 같은 숫자 경로가 유지된다고 보장하지 않습니다. 가능하면 0.3.1.3 같은 숫자 경로 대신, 안정적으로 유지되는 Companion Tree/variables/internal/instance_warns 같은 텍스트 경로를 사용하십시오. 큰 변경 후 Companion을 완전히 재시작하면 숫자 경로를 안정화하는 데 도움이 될 수 있습니다.

4.11 RossTalk 제어

RossTalk는 Ross Video의 간단한 텍스트 기반 제어 프로토콜입니다. TCP 포트 7788로 RossTalk 명령을 보내 Companion을 실행할 수 있습니다. 이 포트는 고정되어 변경할 수 없습니다. 먼저 설정(Settings)에서 수신 기능을 켜야 합니다.

주의

RossTalk 명령 끝에는 줄바꿈 문자, 즉 \n (0x0A) 또는 \r\n (0x0D, 0x0A)을 붙여야 합니다.

명령

Companion은 CC(Custom Command) 명령을 수신합니다. 뒤따르는 값으로 누를 버튼을 지정합니다.

- CC <page>/<row>/<column>
  격자 위치의 버튼을 눌렀다가 놓기
- CC <page>:<button> (사용 중단 예정)
  이전 방식의 버튼 번호로 버튼을 눌렀다가 놓기

<page>:<button> 형식은 8열 격자의 이전 버튼 번호 1-32를 사용합니다. 버튼 1은 0행 / 0열, 버튼 9는 1행 / 0열이며 나머지도 같은 규칙입니다. 이 형식은 이전 버전과의 호환을 위해서만 유지됩니다. 새로 설정할 때는 <page>/<row>/<column> 형식을 사용하십시오.

예제

2페이지 3행 1열 버튼을 눌렀다가 놓기
CC 2/3/1

2페이지의 버튼 5를 눌렀다가 놓기
CC 2:5

다른 소프트웨어에서 RossTalk 사용

RossTalk를 지원하는 여러 장치와 애플리케이션에는 CC 접두사를 자동으로 붙이는 전용 "Custom Command" 입력란이 있습니다. 이 경우 CC 2/3/1을 **입력하지 말고** 2/3/1처럼 명령 본문만 입력하십시오.

ProPresenter

ProPresenter는 입력 내용 앞에 CC 를 자동으로 붙이므로 명령 본문만 입력해야 합니다.

1. ProPresenter에서 Companion의 IP 주소와 포트 7788을 사용하는 RossTalk 장치를 추가합니다.
2. 트리거의 명령에는 2/3/1(2페이지, 3행, 1열)처럼 버튼 위치만 입력합니다.
3. CC 접두사를 **붙이지 마십시오**. CC 2/3/1을 입력하면 ProPresenter가 CC CC 2/3/1을 전송합니다.

전체 설정 절차는 Renewed Vision의 RossTalk 장치 가이드를 참조하십시오.

4.12 Satellite 프로토콜

Satellite 프로토콜은 Stream Deck 등의 원격 제어장치를 네트워크를 통해 중앙 Companion 인스턴스에 연결합니다. 장치가 Companion 실행 컴퓨터와 다른 컴퓨터에 연결되어 있거나 다른 방에 있을 때 유용합니다.

일반적으로 원격 컴퓨터에서 Companion Satellite 앱을 실행하여 사용합니다. 앱은 연결된 제어장치를 TCP 또는 WebSockets로 Companion에 전달합니다.

설정

1. 원격 컴퓨터에 **Companion Satellite**를 설치하고 실행합니다.
2. Companion의 설정에서 **Satellite TCP Listen Port**를 확인합니다. 기본값은 16622입니다.
3. Companion Satellite에 Companion 실행 컴퓨터의 IP 주소를 지정합니다.
4. 연결되면 원격 제어장치가 직접 연결한 장치처럼 Companion의 제어장치(Surfaces) 페이지에 표시됩니다.

도움말

Companion이 방화벽 뒤나 Docker에서 실행된다면 TCP 포트 16622와 WebSockets용 16623이 Companion 호스트로 전달되도록 설정하십시오.

개발자 문서

사용자 제작 장치를 연결하는 등 자체 Satellite 클라이언트를 개발하려면 개발자 문서의 전체 프로토콜 규격을 참조하십시오.

5 웹 버튼과 에뮬레이터

5.1 에뮬레이터

에뮬레이터(Emulator) 는 브라우저에 간단한 버튼 그리드를 표시합니다. 실제 Stream Deck처럼 Companion과 연동하므로 설정을 시험하거나 적은 비용으로 다른 사용자에게 조작 버튼을 제공할 때 사용할 수 있습니다.

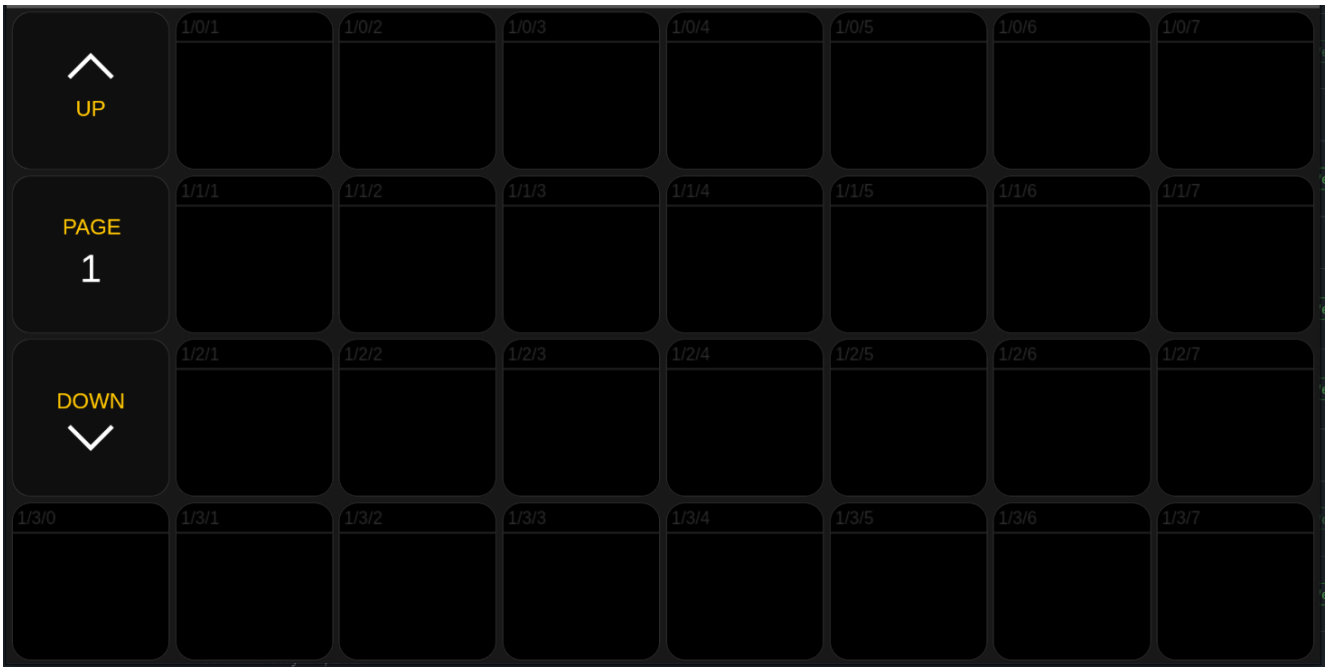


그림 5-1. 에뮬레이터

Companion을 새로 설치하면 기본적으로 에뮬레이터가 생성되지 않습니다. Companion 3.0부터 여러 에뮬레이터를 만들 수 있습니다. 하나의 에뮬레이터를 여러 곳에서 동시에 열면 모두 같은 Companion 페이지를 공유합니다.
여러 에뮬레이터를 사용하면 사용자마다 서로 다른 페이지를 동시에 볼 수 있습니다.

키보드 단축키로 에뮬레이터를 조작하고 버튼을 누를 수도 있습니다. 사용 방법은 에뮬레이터 선택 페이지 상단에 있습니다.

에뮬레이터별 추가 설정은 제어 장치(Surfaces) 페이지에서 할 수 있습니다.

5.2 웹 버튼

웹 버튼(Web buttons) 화면은 모든 페이지의 버튼을 한 화면에 표시합니다. 태블릿의 웹 브라우저로 Companion을 조작할 때 유용합니다.

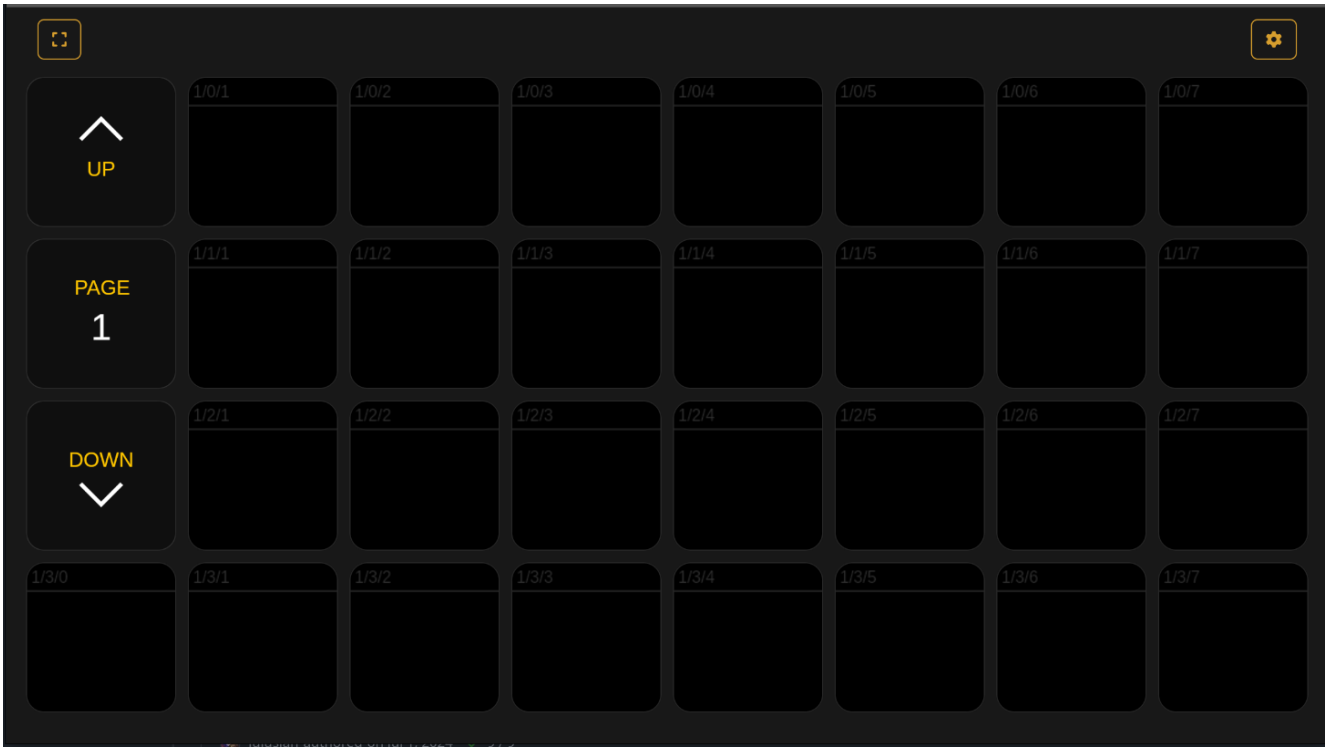


그림 5-2. 웹 버튼

참고

웹 버튼 화면은 Companion에서 **제어 장치(Surface)** 로 인식되지 않으므로, 버튼에 설정한 페이지 전환 동작을 따르지 않습니다.

이 페이지에는 다양한 설정 옵션이 있습니다. 오른쪽 위의 톱니바퀴를 클릭해 설정 패널을 엽니다.
선택한 설정은 URL 매개변수로 저장되므로 설정을 마친 뒤 URL을 북마크에 저장하십시오.

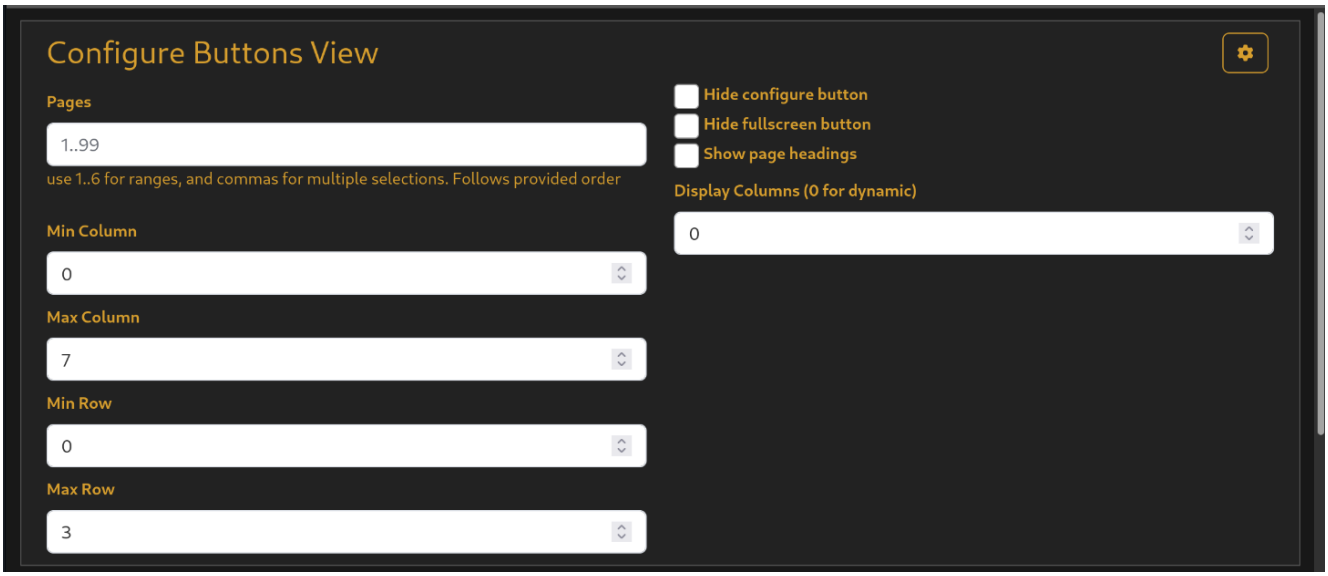


그림 5-3. 웹 버튼 설정

이전 Companion에는 웹과 태블릿용 버튼 화면이 여러 가지 있었습니다. Companion 내부 구조가 바뀌면서 이 화면들을 새로 작성해 기능이 더 풍부한 하나의 화면으로 통합했습니다.

6 지원 제어 장치

Companion은 원격 제어 프로토콜뿐 아니라 다양한 물리적·가상 입력 장치인 제어 장치(Surface)를 지원합니다. 제어 장치는 버튼, 인코더, 디스플레이 등 Companion의 액션에 연결할 수 있는 여러 조작 요소를 제공합니다.

장치마다 제공하는 기능과 제한이 다를 수 있습니다. 모델별 정보, 설정 요령, 알려진 제한 사항은 각 제어 장치의 안내 페이지를 참고하십시오.

제어 장치 모듈과 통합 기능

Companion 4.3부터는 연결(Connection)과 같은 모듈 시스템으로 제어 장치를 지원합니다. 따라서 새 모델이 출시되거나 기존 장치의 버그가 수정되어도 Companion 전체를 업데이트하지 않고 해당 제어 장치 모듈만 업데이트할 수 있습니다.

제어 장치 통합 모듈은 제어 장치 페이지의 오른쪽 패널에서 설정합니다. 브라우저에 패널이 하나만 보이면 파란색 Show Settings(설정 표시) 버튼을 클릭해 설정 패널을 여십시오.

오른쪽 패널에서 빨간색 Add Surface Integration(제어 장치 통합 추가) 버튼을 클릭합니다.

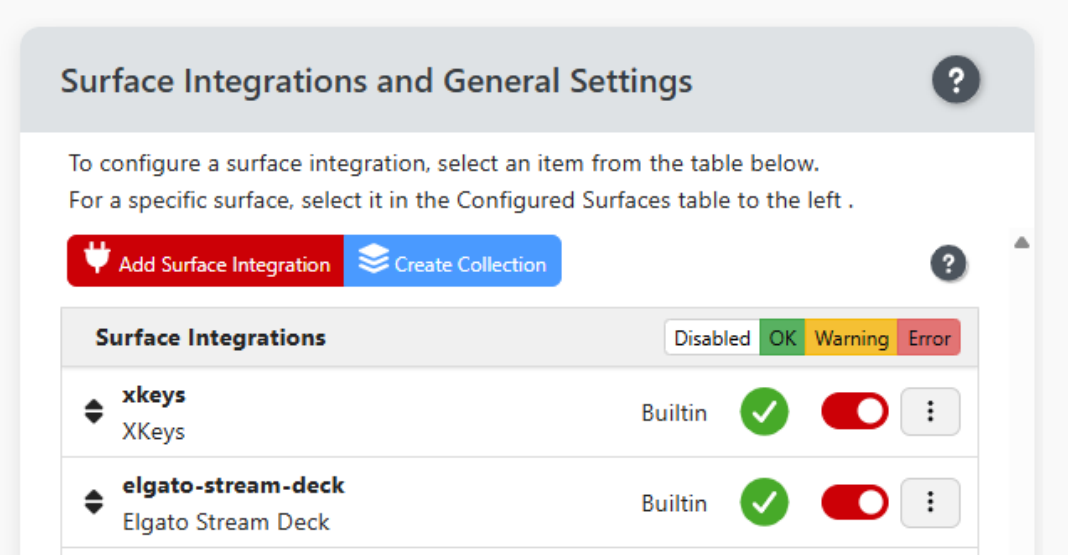


그림 6-1. 제어 장치 통합 패널

All Available 버튼이 파란색이 아니면 클릭해 아직 설치하지 않은 모듈을 표시합니다. 버튼이 All Available처럼 표시되면 사용 가능한 모든 제어 장치 모듈이 나열됩니다.

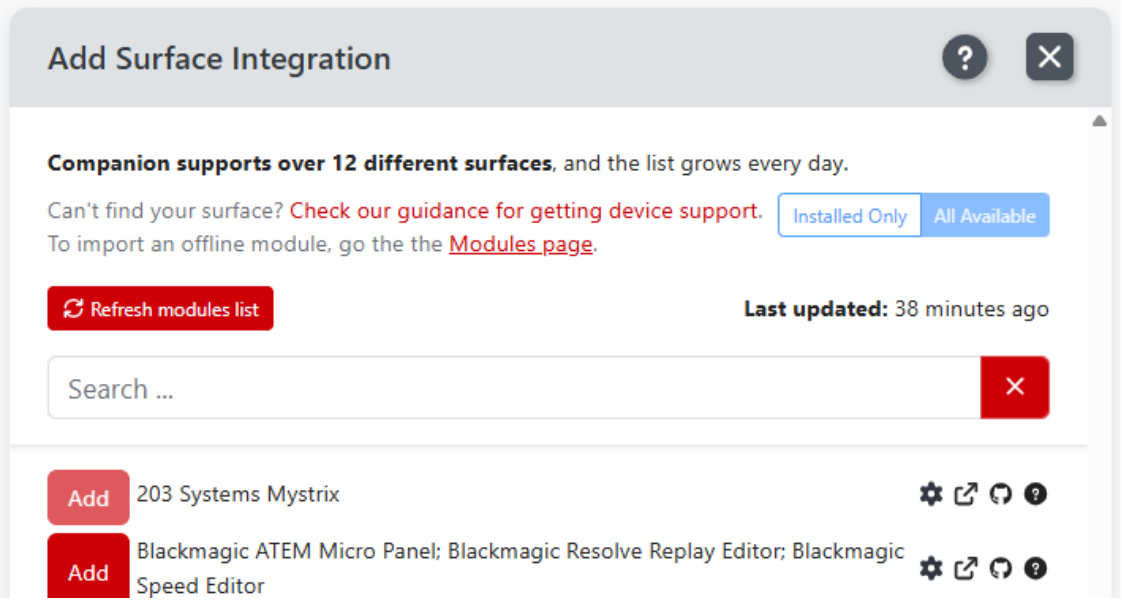


그림 6-2. 제어 장치 통합 추가 패널

이후 제어 장치 모듈을 설치, 설정, 업데이트하는 방법은 연결 모듈과 같습니다.

모듈 페이지에서도 제어 장치 모듈을 추가할 수 있으며, 오프라인 모듈을 가져올 수도 있습니다.

사용 가능한 제어 장치 모듈과 지원 모델의 전체 목록은 Bitfocus 개발자 사이트에서도 확인할 수 있습니다.

제어 장치 공통 설정

제어 장치 공통 설정도 제어 장치 페이지의 오른쪽 패널에 있습니다. 브라우저에 패널이 하나만 보이면 파란색 Show Settings(설정 표시) 버튼을 클릭해 설정 패널을 여십시오.

자세한 내용은 제어 장치 설정 또는 제어 장치 구성 안내를 참고하십시오.

6.1 Companion Satellite

Companion Satellite는 별도의 컴퓨터에서 실행되는 가벼운 애플리케이션입니다. 해당 컴퓨터에 연결된 제어 장치(Surface)를 네트워크를 통해 중앙의 Companion 인스턴스에 전달합니다.

다음과 같은 경우에 유용합니다.

- Stream Deck이 Companion 서버와 다른 책상이나 다른 방에 있을 때
- 서버 또는 USB를 직접 사용할 수 없는 Docker에서 실행 중인 Companion에 제어 장치를 연결할 때
- 여러 운영자가 각자의 Stream Deck을 하나의 Companion 설정으로 사용할 때

작동 방식

Companion Satellite는 TCP(기본 포트 16622) 또는 WebSockets(기본 포트 16623)로 Companion에 연결합니다. 연결되면 Satellite 컴퓨터에 연결된 제어 장치가 Companion의 제어 장치(Surfaces) 페이지에 나타나며, 로컬 장치와 같은 방식으로 설정할 수 있습니다.

시작하기

1. Bitfocus 웹사이트에서 Companion Satellite를 내려받아 설치합니다.
2. 원격 컴퓨터에서 Satellite를 실행하고 Companion 서버의 IP 주소를 입력합니다.
3. Companion에서 **제어 장치(Surfaces)** 페이지를 엽니다. Satellite가 연결되면 원격 제어 장치가 자동으로 나타납니다.

참고

Satellite가 연결되지 않으면 Companion 호스트의 16622 및 16623 포트에 접근할 수 있는지 확인하십시오.
수신 포트는 설정(Settings)에서 확인할 수 있습니다.

설정

목록에 나타난 제어 장치는 다른 장치와 같은 방식으로 설정합니다. 사용할 수 있는 설정은 제어 장치 안내를 참고하십시오.

참고

Companion Satellite가 지원하는 제어 장치는 Companion에 직접 연결할 때보다 제한적입니다.
현재 지원하는 장치 목록은 Satellite 릴리스 노트를 확인하십시오.

7 보안

Companion에는 관리 화면과 제어장치(Surface)에 대한 가벼운 무단 접근을 제한하는 여러 기능이 있습니다.
이 기능들만으로 설치 환경의 보안을 보장할 수는 없지만, 권한 없이 살펴보거나 조작하는 일을 줄이는 데 도움이 됩니다.

관리 화면에는 보안 설정을 한곳에 모아 둔 페이지가 없으므로, 이 절에서 관련 기능을 함께 안내합니다.

1. 관리자 비밀번호, 설정(Settings) > 고급(Advanced) > Admin UI Password: 켜면 모든 설정 페이지를 열 때 비밀번호를 요구합니다.
2. 제어장치 잠금 화면, 제어장치(Surfaces) 페이지의 PIN Lockout: 일정 시간이 지나면 연결된 모든 제어장치를 잠그고, 잠금 해제 시 PIN을 요구합니다.
3. 페이지 권한, Surfaces 탭: 장치나 그룹에 적용하면 해당 장치 또는 그룹이 접근할 수 있는 페이지를 제한할 수 있습니다.
4. 원격 접근, 설정 > 프로토콜(Protocols): 원격 컴퓨터나 장치가 Companion에 연결할 수 있는지 설정합니다.
5. 위험한 기능: Companion 5.0부터 셸 명령 실행과 원격 클라이언트가 보낸 모듈 설치는 **기본으로 비활성화**됩니다. 이 기능은 Companion이 실행되는 컴퓨터에서 코드를 실행하도록 허용하므로, 꼭 필요하고 네트워크를 신뢰할 수 있을 때만 켜십시오.
6. REST API 키, 설정 > 프로토콜 > REST API: REST API는 기본으로 꺼져 있으며, 켜면 모든 요청에 API 키가 필요합니다. 스크립트나 시스템마다 필요한 권한 범위만 가진 별도의 키를 발급하고, 더 이상 사용하지 않는 키는 폐기하십시오. **최근 사용(Last used)** 열에서 사용하지 않는 키를 확인할 수 있습니다.
7. Prometheus 지표, 설정 > 프로토콜 > Prometheus Metrics: 기본으로 꺼져 있으며, 켜도 수집 토큰(scrape token)이 있어야 지표를 읽을 수 있습니다.

관리자 비밀번호는 관리 화면만 보호합니다. 원격 제어 프로토콜, REST API, 지표 엔드포인트는 보호하지 않으며,
각각 위에서 설명한 별도의 설정을 사용합니다.

부록 · 원문과 편집 기준

이 설명서는 Bitfocus Companion 5.1.1 안정 릴리스와 공식 v5.1 사용자 가이드를 기준으로 번역·편집한 한국어판입니다.
원문 확인일은 2026년 10월 29일입니다.

공식 사용자 가이드의 본문 57개를 수록했습니다. 개발자 문서와 개별 장치 모듈의 전용 설명서는 별도 자료입니다.
프로그램 화면 이미지는 원문을 사용하며, 본문·표·그림 설명을 한국어로 편집했습니다.

UI 항목은 화면에서 찾을 수 있도록 필요한 위치에 영어를 함께 표시했습니다. 제품명, 모델명, 프로토콜, 기술 고유 명칭, 명령어와 코드의 식별자는 원문을 유지했습니다.

공식 원문: Companion v5.1 User Guide
릴리스: Bitfocus Companion v5.1.1

원문 문서의 MIT License에 따라 한국어 번역과 편집을 적용했습니다. 아래에 원문의 저작권 및 허가 고지를 보존합니다.
이 한국어판은 Bitfocus가 발행한 공식 번역본이 아닙니다.

MIT License · 원문 고지

Copyright (c) 2018 Bitfocus AS - William Viker & Håkon Nessjøen, http://www.bitfocus.io/

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.