=========================================

코드 검토 가이드 라인 및 모범 사례

=========================================

SmartApp 또는 Device Handler를 제출하기 전에 본인의 코드가 이 문서에 기재 된 지침을 준수하는지 확인해야합니다.

이 가이드 라인을 준수하지 않는 코드는 거절될 수 있습니다.

이 문서는 SmartThings 개발을 위한 모범 사례 모음집이기도 합니다.

----

일반적인 규칙

-------

코드는 읽기 쉬워야합니다.

^^^^^^^^^^^^^^^^^^^^^^^

코드는 기계가 실행하지만, 사람이 읽습니다.

가독성은 주관적 일 수 있지만 따라야 할 몇 가지 일반적인 지침이 있습니다:

- 의미 있는 변수명 및 메소드 이름을 사용하세요.

- :참고:`review\_guidelines\_dry`

- :참고:`review\_guidelines\_methods`

- :참고:`review\_guidelines\_comments`

.. \_review\_guidelines\_dry:

중복하지 마세요

^^^^^^^^^^^^^^^^^^^^^

`DRY 법칙 <https://en.wikipedia.org/wiki/Don%27t\_repeat\_yourself>`\_\_ (don't repeat yourself 중복하지 말라)을 따라주세요.

코드를 복사/붙여넣기 하지 마세요 – 자주 쓰이는 코드를 공유 유틸리티 메소드로 빼주세요.

.. \_review\_guidelines\_methods:

메소드는 한가지 목적만 가져야합니다

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

메소드는 한가지 목적만을 충족시켜야하며 간결해야합니다.

메서드의 정의가 표준 컴퓨터 화면을 넘어간다면, 너무 길다는 겁니다.

코드를 유틸리티 메서드로 분리할 수 있는지 알아보세요.

예를 들어, 큰 사이즈의 HTTP 응답을 즉시마다 분석하는 메서드는 길어질 수 있으니, 이 작업을 호출할 수 있는 여러 메서드로 분리시키세요.

이렇게 하면, 코드를 더 쉽게 이해할 수 있게 되며 더 나은 `관심사의 분리 <https://en.wikipedia.org/wiki/Separation\_of\_concerns>`\_\_를 보장합니다.

사용하지 않는 코드를 제출하지 마세요

^^^^^^^^^^^^^^^^^^^^^^^^^

사용하지 않거나 주석처리 된 코드는 제출하기 전에 지워 주셔야합니다.

모욕적인 말, 모독적인 언어 또는 비방하는 언어를 사용하지 마세요

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

따로 설명이 필요 없겠지만 – 언어는 깔끔하고 전문적이어야합니다.

.. \_review\_guidelines\_comments:

적절한 주석을 달아주세요

^^^^^^^^^^^^^^^^^^^^^

주석은 적절하게 사용될 때 코드에 명확성과 내용을 더할 수 있습니다.

지나치게 많이 사용되면 코드가 복잡해지며 쓸모 없어집니다.

따라야 할 몇 가지 지침이 있습니다:

- 일반적으로 코드가 일반적인 생각과 다르게 실행될 때 주석이 필요합니다.

- Device Handler 사용자 커맨드 및 속성에는 용도, 매개 변수 및 예외 조건 (적용 가능한 경우)을 설명하는 주석이 있어야합니다.

- 중요한 메소드는 그 메소드가 하는 일, 반환형, 예외 조건 및 매개 변수를 설명하는 주석을 함께 작성해야합니다. `JavaDoc 형식 주석 <https://en.wikipedia.org/wiki/Javadoc#Overview\_of\_Javadoc>`\_\_을 사용할 수 있지만 소스에서 문서를 생성 할 수 있는 도구는 없습니다.

- 주석은 가치를 더해야합니다 - 코드의 모든 행에 주석을 더하면 코드가 혼란스러워질 뿐더러 불필요한 일입니다.

메소드에 주석을 적절히 작성한 예시입니다:

.. code-block:: groovy

def capabilityCommands = getDeviceCapabilityCommands(device.capabilities)

/\*\*

\* Builds a map of capability names to their supported commands.

\*

\* @param a list of Capabilities.

\* @return a map of device capability -> supported commands.

\*/

def getDeviceCapabilityCommands(deviceCapabilities) {

def map = [:]

deviceCapabilities.collect {

map[it.name] = it.commands.collect{ it.name.toString() }

}

return map

}

Here's an example of an in-line code comment explaining why the code is checking if a percentage value is within a certain hard-coded range:

다음은 퍼센트 값이 해당범위 안에 있는지 확인하는 이유를 설명한 인라인 주석입니다:

.. code-block:: groovy

log.trace "stopDimmersHandler evt: ${evt.value}"

def percentComplete = completionPercentage()

// 많은 경우에 우리가 가장 먼저 하는 일은 조명을 켜거나 끄는 것입니다.

     // 그러니 시작하자 마자 멈추지 않도록 해야합니다.

if (percentComplete > 2 && percentComplete < 98) {

...

}

부적절한 주석의 예는 다음과 같습니다.

코드만 읽어도 분명한 내용을 주석이 단순히 반복하고 있습니다: 가치가 더해지지 않고 있습니다.

.. code-block:: groovy

// 모든 자식을 받아와라

def children = pollChildren()

// 모든 자식을 돌아라

children.each {child ->

// 각 자식을 로그로 띄워라

log.debug "child: $child"

}

모든 ``if()``와 ``switch()`` 구문을 확인하십시오

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

``if ()``또는``switch ()``블록이 모든 예상 입력을 처리하는지 확인하십시오.

특정 조건을 처리하는 것을 잊어버리면 예기치 못한 논리 오류가 발생할 수 있습니다.

Also, every ``switch()`` statement should have a ``default:`` case statement to handle any cases where there is no match.

또한 모든``switch ()``문은 일치하는 조건이 없는 경우를 처리하기 위해``default :`` 조건문을 가져야합니다

가정을 확인하세요

^^^^^^^^^^^^^^^^^^

If a method operates on some input, it should handle all possible input values, including any differences if the method is called from a parent or child SmartApp or Device Handler.

메소드가 일부 입력에 대해 작동하는 경우 메소드가 상위 또는 하위 SmartApp 또는 Device Handler에서 호출되는 경우의 차이점을 포함하는 모든 입력 값을 처리할 수 있어야합니다.

일관된 반환 값 사용

^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Groovy is a dynamically typed language.

That's great for a lot of things, but it's a sharp knife - highly effective, yet also easy to cut yourself accidentally.

Groovy는 동적 유형 지정 언어입니다.

그것은 많은 일을 하는 데 적합하지만 날카로운 칼입니다. 매우 효과적이면서 실수로 쉽게 자를 수 있습니다.

A method should return a single type of data, regardless of if the method signature is typed or not.

For example, don't do something like this:

메소드 서명이 입력되었는지 여부에 관계없이 메소드는 단일 유형의 데이터를 리턴해야합니다.

예를 들어, 다음과 같이하지 마십시오.

.. code-block:: groovy

def getSomeResult(input) {

if (input == "option1") {

return true

}

if (input == "option2") {

return false

}

return [name: "someAttribute", value: input]

}

The example above fails to return a consistent data type.

Calling clients of this code have to accommodate both a boolean and map return values.

Instead, methods should always return the same data type.

위의 예제는 일관된 데이터 형식을 반환하지 않습니다.

이 코드를 호출하는 클라이언트는 부울 값과 맵 반환 값을 모두 수용해야합니다.

대신 메서드는 항상 동일한 데이터 형식을 반환해야합니다.

.. note::

In certain cases, it \*may\* make sense for a method to return different types.

Such cases are the exception, and the different types returned, and under what circumstances, should be documented in the method's comments.

특정의 경우, 메소드가 다른 형태를 돌려주는 것은 의미가있는 일이 있습니다.

     이러한 경우는 예외이며 반환되는 다른 유형과 어떤 상황에서 메소드의 주석에 문서화되어야합니다.

Be careful indexing into arrays

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

배열에주의를 기울여 색인 작성

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

When parsing data, pay attention to arrays if you use them.

Do not index into arrays directly without making sure that the array actually has enough elements.

데이터를 파싱 할 때 배열을 사용한다면주의를 기울이십시오.

배열에 실제로 충분한 요소가 있는지 확인하지 않고 직접 배열을 인덱싱하지 마십시오.

Consider the following code that splits a string on the ``":"`` character, and returns the value after the ``":"``:

`` ":"`문자에 문자열을 분할하고`` ":"``다음에 값을 반환하는 다음 코드를 생각해보십시오.

.. code-block:: groovy

def getSplitString(input) {

return input.split(":")[1]

}

// -> "123"

getSplitString("abc:123")

// -> ArrayIndexOutOfBounds exception!

getSplitString("abc:")

Because ``getSplitString()`` does not verify that the result of ``split()`` split has more than one element, we get an ``ArrayIndexOutOfBounds`` exception when trying to access the second item in the parsed result.

In cases like this, make sure your code verifies the array contains the item:

``getSplitString ()``은``split ()`split의 결과가 하나 이상의 원소를 가지고 있는지를 검증하지 않기 때문에, 파싱 된 결과에서 두번째 항목에 접근하려 할 때``ArrayIndexOutOfBounds`` 예외가 발생합니다.

이와 같은 경우 코드가 배열에 항목이 있는지 확인합니다.

.. code-block:: groovy

def getSplitString(input) {

def splitted = input?.split(":")

if (splitted?.size() == 2) {

return splitted[1]

} else {

return null

}

}

Use the Elvis operator correctly

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Elvis 연산자를 올바르게 사용하십시오.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Groovy supports the Elvis operator, which allows us write more concise conditional expressions than otherwise possible.

However, we need to understand :ref:`Groovy truth <review\_guidelines\_groovy\_truth>` to use it effectively.

Groovy는 Elvis 연산자를 지원합니다. Elvis 연산자를 사용하면 가능한 경우보다 간결한 조건식을 작성할 수 있습니다.

그러나이를 효과적으로 사용하려면 ref :`Groovy truth <review\_guidelines\_groovy\_truth> '를 이해해야합니다.

Consider this example that attempts to set the variable ``bulbLevel`` to ``100`` if it is not already set:

변수``bulbLevel``을 아직 설정하지 않은 경우``100``을 설정하려고 시도하는이 예제를 생각해보십시오 :

.. code-block:: groovy

def bulbLevel = settings.level ?: 100

But what happens if ``settings.level`` is ``0`` in the example above? \*\*Because Groovy considers zero as false, we've set\*\* ``bulbLevel`` \*\*to\*\* ``100`` \*\*!\*\*

그러나 위의 예제에서``settings.level``이``0``이라면 어떻게 될까요? \*\* Groovy는 0을 false로 간주하기 때문에 \*\*``bulbLevel`` \*\*을 \*\*``100`` \*\*! \*\*으로 설정했습니다! \*\*! \*\*

The above expression should be rewritten as:

위의 표현식은 다음과 같이 다시 작성되어야합니다.

.. code-block:: groovy

def bulbLevel = settings.level == null ?: 100

Handle null values

^^^^^^^^^^^^^^^^^^

Null 값 처리

^^^^^^^^^^^^^^^^^^^^

.. important::

.. important :: 중요한

NullPointerExceptions are one of the most frequently occurring exceptions on the SmartThings platform - take care to avoid them!

     NullPointerExceptions은 SmartThings 플랫폼에서 가장 자주 발생하는 예외 중 하나입니다.이를 피하기 위해주의하십시오!

This is \*very\* common in LAN and SSDP interactions, so always double check that code.

   이것은 LAN과 SSDP 상호 작용에서 \* 매우 \* 공통적이므로 항상이 코드를 두 번 확인하십시오.

A ``NullPointerException`` will terminate the SmartApp or Device Handler execution, but can be avoided easily with the `safe navigation <http://groovy-lang.org/operators.html#\_safe\_navigation\_operator>`\_\_ (``?``) operator.

Any code that may encounter a ``null`` value should anticipate and handle this.

``NullPointerException``은 SmartApp 또는 Device Handler의 실행을 종료 시키지만`안전한 탐색 <http://groovy-lang.org/operators.html#\_safe\_navigation\_operator>`\_\_ (``?`` ) 운영자.

``null`` 값을 만나게 될 모든 코드는 이것을 예상하고 처리해야합니다.

The examples below show a few common scenarios in which ``null`` is possible, and how to deal with it using the ``?`` operator:

아래 예제는``null``이 가능한 몇 가지 일반적인 시나리오와``?``연산자를 사용하여 그것을 처리하는 방법을 보여줍니다 :

.. code-block:: groovy

// if the LAN event does not have headers, or a "content-type" header,

// don't blow up with a NullPointerException!

// LAN 이벤트에 헤더 또는 "content-type"헤더가없는 경우,

     // NullPointerException을 날려 버리지 마라!

if (lanEvent.headers?."content-type"?.contains("xml")) { ... }

.. code-block:: groovy

// if a location does not have any modes, statement simply returns null

// but does not throw a NullPointerException

// 위치에 모드가없는 경우 명령문은 null을 반환합니다.

     // 그러나 NullPointerException을 throw하지 않습니다.

if (location.modes?.find{it.name == newMode}) { ... }

.. \_review\_guidelines\_groovy\_truth:

Use Groovy truth correctly

^^^^^^^^^^^^^^^^^^^^^^^^^^

그루비 진리를 올바르게 사용하십시오.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Be aware of, and ensure your code is consistent with, what Groovy considers true and false.

Groovy truth is documented `here <http://groovy-lang.org/semantics.html#Groovy-Truth>`\_\_.

Groovy가 true 또는 false로 간주하는 코드를 인식하고 코드가 일관성을 유지하는지 확인하십시오.

그루비 진리는`여기 http://groovy-lang.org/semantics.html#Groovy-Truth>`\_\_에 문서화되어있다.

Here are some gotchas to be aware of:

알고 있어야 할 몇 가지 문제점이 있습니다

- Empty strings are considered ``false``; non-empty strings are considered ``true``.

- Empty maps and lists are considered ``false``; non-empty maps and lists are considered ``true``.

- Zero is considered ``false``; non-zero numbers are considered ``true``.

- 빈 문자열은``거짓``으로 간주됩니다; 비어 있지 않은 문자열은 "참"으로 간주됩니다.

- 빈지도와 목록은 "거짓"으로 간주됩니다. 비어 있지 않은지도와 목록은 "사실"로 간주됩니다.

- 0은 "거짓"으로 간주됩니다. 0이 아닌 숫자는 "참"으로 간주됩니다.

Consider the following example that verifies that a number is between 0 and 100:

숫자가 0과 100 사이인지 확인하는 다음 예제를 고려하십시오

.. code-block:: groovy

def verifyLevel(level) {

if (!level) {

return false

} else {

return (level >= 0 && level <= 100)

}

}

If we call ``verifyLevel(0)``, the result is ``false``, because ``0`` is treated as false by Groovy.

Instead, it should be written as:

Groovy가``0``을 false로 처리하기 때문에``verifyLevel (0)``을 호출하면 결과는``false``입니다.

대신 다음과 같이 작성해야합니다.

.. code-block:: groovy

def verifyLevel(level) {

return (level instanceof Number && level >= 0 && level <= 100)

}

This can be a common source of errors; make sure you understand and use Groovy truth appropriately.

이것은 일반적인 오류의 원인이 될 수 있습니다. Groovy 진리를 이해하고 적절하게 사용하는지 확인하십시오.

----

Using State

-----------

주 사용

-----------

``state`` is not an unbounded database

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

``state``는 무한대의 데이터베이스가 아닙니다.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

``state`` (SmartApps and Device Handlers) and ``atomicState`` (SmartApps only) are provided to persist small amounts of data across executions.

Do not think of state as a virtually unlimited database for your app.

``state`` (SmartApps and Device Handlers)와``atomicState`` (SmartApps 만)는 실행을 통해 소량의 데이터를 유지하기 위해 제공됩니다.

상태를 앱의 거의 무제한 데이터베이스라고 생각하지 마십시오.

The amount of data that can be stored in state is :ref:`limited <state\_size\_limit>`.

Avoid code that adds items to ``state`` regularly (perhaps in response to Events or schedules), but does not remove items.

상태에 저장할 수있는 데이터의 양은 ref :`limited <state\_size\_limit>`입니다.

주기적으로 이벤트를 추가하거나 항목을 제거하지 않는 코드는 피하십시오.

Understand how ``state`` works

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

``국가``의 작동 방식 이해

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Remember that when using ``state``, the :ref:`results are not persisted until the app is done executing <state\_how\_it\_works>`.

This can have unintended consequences, such as state values being overridden by another concurrently executing instance of the SmartApp.

``state``를 사용할 때 : ref :`결과는 앱이 <state\_how\_it\_works>를 실행하기 전까지 지속되지 않습니다.

동시에 실행되는 다른 SmartApp 인스턴스가 상태 값을 대체하는 경우와 같이 의도하지 않은 결과가 발생할 수 있습니다.

Understand when to use ``atomicState`` vs. ``state``

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

``atomicState``와``state``를 사용할시기를 이해하십시오.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ ^^

Understand the :ref:`difference <choosing\_between\_state\_atomicState>` between ``atomicState`` and ``state``, make sure you use the correct one for your needs, and avoid using both in the same SmartApp.

``atomicState``와``state`` 사이의 : ref :`difference <choose\_between\_state\_atomicState>를 이해하고, 필요에 맞게 올바른 것을 사용하고 동일한 SmartApp에서 둘 다 사용하는 것을 피하십시오.

Take care when storing collections in ``atomicState``

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

콜렉션을``atomicState``에 저장할 때주의하십시오.

Modifying collections in Atomic State does not work as it does with State.

:ref:`Read the documentation <atomic\_state\_collections>` to understand how to best work with collections stored in Atomic State.

원자 상태에서 컬렉션 수정은 상태와 마찬가지로 작동하지 않습니다.

: ref :`Atomic State에 저장된 콜렉션에 대한 최선의 작업 방법을 이해하려면 <atomic\_state\_collections> 문서를 읽으십시오.

----

Web Services

------------

웹 서비스

------------

Document external HTTP requests

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

외부 HTTP 요청 문서화

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

:ref:`HTTP requests <calling\_web\_services>` to outside services should be documented, explaining the need to make external requests, what data is sent, and how it will be used.

Please also include a comment with a link to the third party's privacy policy, if applicable.

: 외부 요청에 대한 필요성, 전송되는 데이터 및 사용 방법을 설명하는 : ref :`외부 서비스에 대한 HTTP 요청 <call\_web\_services> '을 문서화해야합니다.

해당하는 경우 제 3 자의 개인 정보 취급 방침에 대한 링크가 포함 된 주석도 함께 적어주십시오.

Document any exposed endpoints

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

노출 된 모든 엔드 포인트 문서화

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

If your SmartApp or Device Handler :ref:`exposes any endpoints <web\_services\_mapping\_endpoints>`, add comments that document what the API will be used for, what data may be accessed by those APIs, and where possible, include a link to the privacy policies of any remote services that may access those APIs.

SmartApp 또는 Device Handler : ref :`모든 엔드 포인트 <web\_services\_mapping\_endpoints>를 공개하는 경우 API가 사용될 대상, API에서 액세스 할 수있는 데이터 및 가능한 경우 개인 정보 보호 정책에 대한 링크를 포함하는 설명을 추가하십시오 이러한 API에 액세스 할 수있는 원격 서비스

----

Scheduling

----------

스케줄링

----------

Avoid recurring short schedules

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

되풀이되는 짧은 일정 피하기

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Scheduled and other periodic functions should not execute more often than every five minutes, unless there is a good reason for it, and the reviewers agree.

예약 된 기능과 기타 주기적 기능은 좋은 이유가 있고 검토자가 동의하지 않는 한 5 분마다 실행하지 않아야합니다.

If your code executes more frequently than every five minutes, add a comment to your code explaining why this is necessary.

코드가 5 분마다 실행되는 경우 코드가 필요한 이유를 설명하는 코드를 추가하십시오.

Avoid chained ``runIn()`` calls

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

``runIn ()``호출을 피하십시오

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

:ref:`Do not chain runIn() calls<scheduling\_chained\_run\_in>`.

: ref :`runIn ()을 chaining하지 말고 <scheduling\_chained\_run\_in>`을 호출하십시오.

If for some reason it is necessary, add a comment describing why it is necessary.

어떤 이유로 든 필요한 이유를 설명하는 설명을 추가하십시오.

----

Security considerations

-----------------------

----

보안 고려 사항

-----------------------

Subscriptions should be clear

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

구독은 명확해야합니다.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

It is possible to subscribe to Events using a string variable, so what the SmartApp is subscribing to might be somewhat opaque.

문자열 변수를 사용하여 이벤트를 구독 할 수 있으므로 SmartApp가 구독하는 대상이 다소 불투명 할 수 있습니다.

For example:

예 :

.. code-block:: groovy

def myContactSubscription = "contact.open"

...

subscribe(contact1, myContactSubscription, myContactHandler)

The best practice is to subscribe explicitly to the attribute:

가장 좋은 방법은 속성에 명시 적으로 가입시키는 것입니다.

.. code-block:: groovy

subscribe(contact1, "contact.open", myContactHandler)

However, if the SmartApp must subscribe to a variable (from state, for instance), the reviewer should be able to trace how the variable is set and what the expected attribute will be.

그러나 SmartApp가 변수 (예 : 상태)를 구독해야하는 경우 검토자는 변수가 설정되는 방식과 예상되는 특성이 무엇인지 추적 할 수 있어야합니다.

Subscriptions should be specific

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

구독은 구체적이어야합니다.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Do not create overly-broad subscriptions.

지나치게 광범위한 구독을 만들지 마십시오.

A SmartApp that is subscribed to every location Event will execute excessively, and is rarely necessary.

Instead, create subscriptions specific to the Event you are interested in.

모든 위치 이벤트에 등록 된 SmartApp는 지나치게 실행되며 거의 필요하지 않습니다.

대신 관심있는 일정에 맞는 구독을 만드십시오.

If you're creating a service manager for a LAN-connected device, be sure to :ref:`subscribe to the device search target <lan\_device\_discovery>`.

LAN 연결 장치 용 서비스 관리자를 만드는 경우 : ref :`장치 검색 대상 <lan\_device\_discovery> 구독 '을 반드시 수행하십시오.

Do not use dynamic method execution

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

동적 메서드 실행을 사용하지 마십시오.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

In groovy you can execute functions based on a string, like so:

Groovy에서는 다음과 같이 문자열을 기반으로 함수를 실행할 수 있습니다.

.. code-block:: groovy

object."${mystring}"()

Which can be very handy, but when ``${mystring}`` comes from a HTTP request, outside the SmartThings platform, or from another SmartApp or Device Handler, we need to validate the input.

매우 편리 할 수 있지만, "$ {mystring}"이 HTTP 요청, SmartThings 플랫폼 외부 또는 다른 SmartApp 또는 Device Handler에서 온 경우 입력을 검증해야합니다.

The preferred method of validation is to use a ``switch()`` statement on the input before doing anything with it:

우선적 인 검증 방법은 무엇인가를하기 전에 입력에 대해``switch ()``문을 사용하는 것입니다 :

.. code-block:: groovy

switch(mystring) {

case "cmd1":

object.cmd1()

break

case "cmd2":

object.cmd2()

break

case "cmd3":

object.cmd3()

break

default:

return "ERROR"

}

Do not hard-code SMS messages

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

SMS 메시지를 하드 코딩하지 마십시오.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Notifications should never be sent to a hard-coded number.

They should always use a number provided by the user using the :ref:`contact input <contact\_book>` (even though Contact Book is not enabled, the contact input type is available and contains a fall-back mechanism for non-Contact Book users. Using this future-proofs your SmartApp).

알림은 절대로 하드 코드 번호로 전송하면 안됩니다.

사용자는 : ref :`contact input <contact\_book> '을 사용하여 사용자가 제공 한 번호를 항상 사용해야합니다 (연락처 북을 사용할 수 없거나 연락처 입력 유형을 사용할 수 있으며 연락처 사용자가 아닌 사용자의 폴백 메커니즘 포함) 이 기능을 사용하면 SmartApp가 보증됩니다.

----

Performance

-----------

----

성능

-----------

Do not use busy loops

^^^^^^^^^^^^^^^^^^^^^

사용중인 루프를 사용하지 마십시오.

^^^^^^^^^^^^^^^^^^^^^^^^^

There is no good reason for the code to run busy loops.

Don't do things like this:

코드가 통화 중 루프를 실행할 좋은 이유가 없습니다.

이런 일은하지 마라.

.. code-block:: groovy

def mywait(ms) {

def start = now()

while (now() < start + ms) {

// do nothing, just wait

}

}

The goal of the above code is to delay execution for a number of milliseconds.

This wastes resources and increases the likelihood that the 20 second execution limit will be exceeded.

위 코드의 목적은 수 밀리 초 동안 실행을 지연시키는 것입니다.

이렇게하면 리소스가 낭비되고 20 초 실행 제한을 초과 할 가능성이 높아집니다.

Instead of trying to force a delay in execution, you should :ref:`schedule <smartapp-scheduling>` a future execution of your app.

실행 지연을 강요하는 대신, 다음과 같이해야한다 : 앱의 향후 실행을 참조 :`schedule <smartapp-scheduling>`.

Do not use ``synchronized()``

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

``synchronized ()``를 사용하지 마라.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Using ``synchronized`` incurs a performance overhead, and is highly unlikely to have any effect.

It should not be used.

``synchronized``를 사용하면 성능 오버 헤드가 발생하고 아무런 영향을 미치지 않습니다.

사용해서는 안됩니다.

When a SmartApp or Device Handler executes, it is executing on one of \*n\* available servers assigned for that Location, where \*n\* is variable depending on Location, current load, and other factors.

Concurrent executions of the SmartApp or Device Handler are not guaranteed, or even likely, to be executing on the same server.

Because of this, trying to force synchronous behavior by using ``synchronized`` would only work in the rare occurrence that a concurrent execution happens on the same server, yet it always incurs overhead.

SmartApp 또는 Device Handler가 실행되면 해당 위치에 할당 된 \* n \* 개의 사용 가능한 서버 중 하나에서 실행됩니다. 여기서 \* n \*은 위치, 현재로드 및 기타 요소에 따라 다릅니다.

SmartApp 또는 Device Handler의 동시 실행은 동일한 서버에서 실행되는 것이 보장되지 않거나 가능성이 있습니다.

이 때문에``synchronized``를 사용하여 동기 동작을 강제 실행하려고하면 동일한 서버에서 동시 실행이 발생하는 드문 경우에만 작동하지만 오버 헤드가 항상 발생합니다

----

LAN-specific

------------

----

LAN 관련

------------

Use the device-specific search

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

장치 별 검색 사용

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Service managers for LAN-connected devices should :ref:`subscribe to the device search target <lan\_device\_discovery>` for device discovery.

LAN에 연결된 장치의 서비스 관리자는 다음을 수행해야합니다. : 장치 검색을 위해 장치 검색 대상 <lan\_device\_discovery>에 가입하십시오.

Handle IP change

^^^^^^^^^^^^^^^^

IP 변경 처리

^^^^^^^^^^^^^^^^^^

Service managers for LAN-connected devices should :ref:`handle any IP change <lan\_device\_health>`.

This can happen when the router power cycles and loses its DHCP mappings.

LAN에 연결된 장치의 서비스 관리자는 : ref :`IP 변경 <lan\_device\_health> '을 처리해야합니다.

이는 라우터 전원이 순환되고 DHCP 매핑이 손실 될 때 발생할 수 있습니다.

----

.. \_review\_guidelines\_parent\_child:

Parent-child relationships

--------------------------

부모 - 자녀 관계

--------------------------

Use separate files

^^^^^^^^^^^^^^^^^^

별도의 파일 사용

^^^^^^^^^^^^^^^^^^^^

When using a parent-child relationship, be it a parent SmartApp with child devices, or a parent SmartApp with child SmartApps, the parent and child should exist in separate files.

부모 - 자식 관계를 사용하는 경우 자식 장치가있는 부모 스마트 응용 프로그램이거나 자식 스마트 응용 프로그램이있는 부모 스마트 응용 프로그램이어야합니다. 부모와 자식은 별도의 파일에 있어야합니다.

Putting the parent and child code in the same file leads to file size bloat, makes the code harder to understand, is error-prone, and difficult to debug.

상위 및 하위 코드를 같은 파일에두면 파일 크기가 커지고 코드를 이해하기 어렵게 만들고 오류가 발생하기 쉽고 디버그하기가 어렵습니다.