

# DIAL

## Discovery And Launch protocol specification Version 1.7.1



Copyright © 2014 Netflix, Inc. All rights reserved.

Redistribution and use of the *DIAL Discovery And Launch protocol specification* (the "DIAL Specification"), with or without modification, are permitted provided that the following conditions are met:

- Redistributions of the DIAL Specification must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions of implementations of the DIAL Specification in source code form must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions of implementations of the DIAL Specification in binary form must include the above copyright notice.
- The DIAL mark, the NETFLIX mark and the names of contributors to the DIAL Specification may not be used to endorse or promote specifications, software, products, or any other materials derived from the DIAL Specification without specific prior written permission. The DIAL mark is owned by Netflix and information on licensing the DIAL mark is available at [www.dial-multiscreen.org](http://www.dial-multiscreen.org).

THE DIAL SPECIFICATION IS PROVIDED BY NETFLIX, INC. "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT ARE DISCLAIMED. IN NO EVENT SHALL NETFLIX OR CONTRIBUTORS TO THE DIAL SPECIFICATION BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE DIAL SPECIFICATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

## 1 Introduction

This document describes **DIAL**, a simple protocol for **DI**scovery **And** **L**aunch that enables second-screen applications to discover and launch first-screen applications on first-screen devices.

The goal of this protocol is to enable CE device owners to enjoy seamless integration of phone and tablet applications with their TV-based entertainment experience.

## 2 Terminology

**First-screen:** a TV, Blu-ray player, set-top-box, or similar device

**Second-screen:** a smartphone, tablet, or similar device

**DIAL Server:** a device implementing the server side of the DIAL protocol, usually a first-screen device.

**DIAL Client:** a device that can discover and launch applications on a DIAL server, usually a second-screen device.

### 3 Example use cases

Four different methods are suggested for using DIAL and for establishing communication between second-screen and first-screen applications:

- First-screen service
- Second-screen service
- Cloud service
- Connecting a second-screen application to a running first-screen application

#### 3.1 First-screen service

A Netflix application on an iPhone discovers a Netflix-enabled TV and then launches the Netflix application on the TV to watch a movie. Steps (a), (b), and (c) are covered by DIAL, steps (d) and (e) are Netflix-specific and outside the scope of this document:

- (a) Netflix application on iPhone discovers the DIAL service on the networked TV.
- (b) Netflix application on iPhone uses DIAL to ask TV to launch the Netflix application. The DIAL server MAY pass a URL to the Netflix application, which the application could use to relay information to the DIAL server.
- (c) Netflix application on TV optionally returns additional information to the DIAL server.
- (d) Netflix application on iPhone discovers the Netflix application on TV.
- (e) Netflix application on iPhone and the Netflix application on TV communicate to show the movie.

#### 3.2 Second-screen service

A YouTube application on an Android tablet discovers a YouTube-enabled TV and then launches the YouTube application on the TV to play a video. After the video ends, the TV returns to the previously playing TV show. Steps (a), (b), and (c) are covered by DIAL, step (d) is YouTube-specific and outside the scope of this document:

- (a) YouTube application on tablet discovers DIAL service on the networked TV.
- (b) YouTube application on tablet uses DIAL to ask TV to launch the YouTube application, passing an IP/port corresponding to the YouTube application on the tablet. In addition to the IP/port, the DIAL server MAY also pass a URL to the YouTube application, which the first-screen application could use to relay information to the DIAL server.
- (c) YouTube application on TV optionally returns additional information to the DIAL server.
- (d) YouTube application on TV communicates with the YouTube application on tablet to show the video, then exits back to normal TV UI.

### 3.3 Cloud service

A (fictitious) WebcamX application on an Android phone discovers a WebcamX-enabled TV and then launches the browser-based HTML5 WebcamX application on the TV to display a webcam stream. Steps (a), (b), and (d) are covered by DIAL, steps (c) and (e) are WebcamX-specific and outside the scope of this document:

- (a) WebcamX application on phone discovers DIAL service on the networked TV.
- (b) WebcamX application on phone uses DIAL to ask TV to launch an HTML5-based browser starting with the URL for the WebcamX application, passing a unique token based on a random number. In addition to the random number, the DIAL server MAY also pass a URL to the WebcamX application, which the application could use to relay information to the DIAL server.
- (c) WebcamX application on TV communicates with a cloud-based WebcamX server, making the server aware of the unique token created in step (b).
- (d) WebcamX application on TV optionally returns additional information to the DIAL server.
- (e) WebcamX application on phone communicates with the WebcamX application on TV via a cloud-based WebcamX server, using the unique token to enable the server to correctly route traffic between the two applications.

### 3.4 Connecting a second-screen application to a running first-screen application

A second-screen device discovers a DIAL service on a networked TV. The device sends a request to determine whether a YouTube application is available on the TV, and the DIAL server response indicates that the application is, in fact, already running. The DIAL response – see step (c) – includes the data from step (c) in section 3.2. (Such data is also communicated in step (c) of sections 3.1 and 3.3.) That information enables the device to connect directly to the YouTube application on the TV. Steps (a), (b), and (c) are covered by DIAL; step (d) is YouTube-specific and outside the scope of this document.

- (a) YouTube application on Android phone discovers DIAL service on the networked TV.
- (b) YouTube application on Android phone uses DIAL to request information about the status of the YouTube application on the TV, such as whether the application is installed on the TV and, if so, if it is stopped or running.
- (c) The DIAL server response indicates that the application is already running. The response also contains additional data that enables the YouTube application on the Android phone to communicate with the running instance of the YouTube TV application.
- (d) YouTube application on Android phone connects directly to running instance of the YouTube TV application.

## 4 Protocol overview

The DIAL protocol has two components: **DIAL Service Discovery** and the **DIAL REST Service**.

*DIAL Service Discovery* enables a DIAL client device to discover DIAL servers on its local network segment and obtain access to the DIAL REST Service on those devices.

The *DIAL REST Service* enables a DIAL client to query, launch, and optionally stop applications on a DIAL server device.

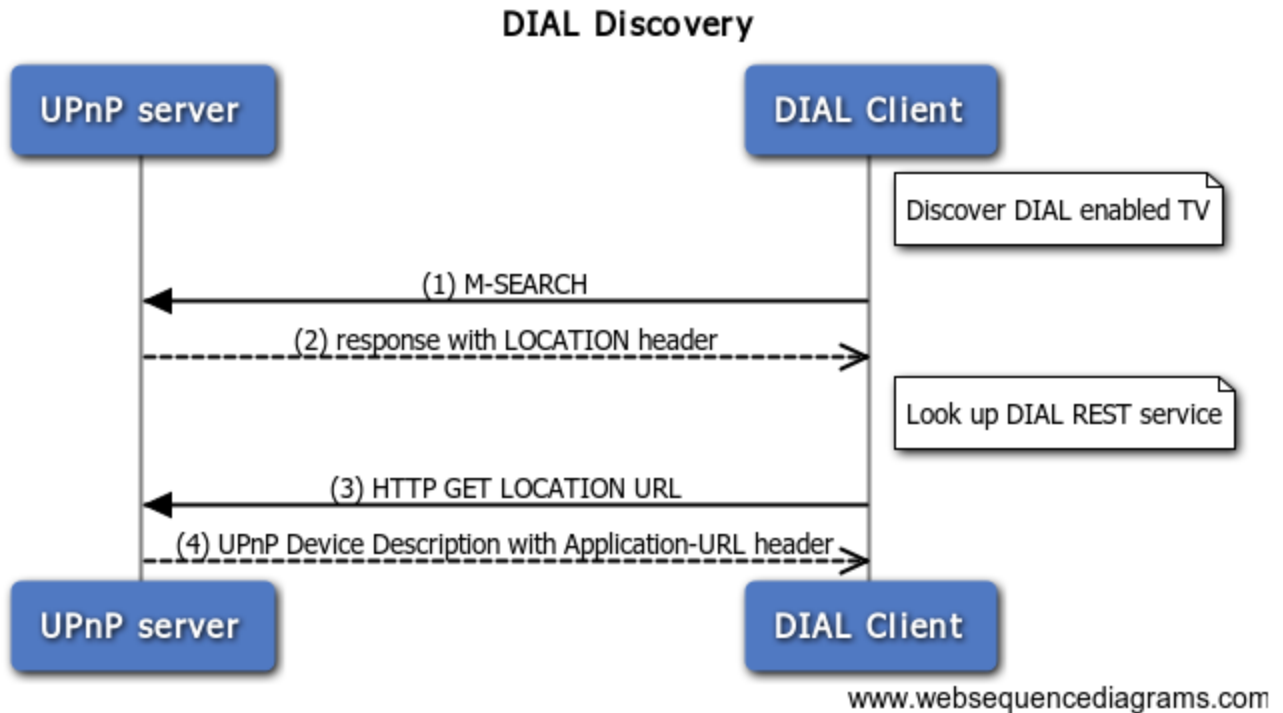
DIAL Service Discovery is achieved using a new Search Target within the SSDP protocol defined by UPnP [1] and an additional header in the response to an HTTP request for the UPnP device description.

The DIAL REST Service is accessed using HTTP [2].

DIAL clients and servers SHALL support the requirements of Section 2.1 of [1]. In particular, HTTP 1.0 SHALL be supported and support of HTTP 1.1 is RECOMMENDED.

## 5 DIAL Service Discovery

The DIAL Service Discovery protocol is based on SSDP (Simple Service Discovery Protocol) version 1.1 as defined in UPnP [1] and HTTP [2] and is illustrated in the following figure:



### 5.1 M-SEARCH request

A DIAL client that wishes to discover DIAL servers SHALL send an M-SEARCH request as defined in Section 1.3.2 of [1] over UDP to the IPv4 multicast address 239.255.255.250 and UDP port 1900 including the Search Target header (ST) with the following value defined by this specification:

```
urn:dial-multiscreen-org:service:dial:1
```



## 5.2 M-SEARCH response

An SSDP/UPnP server receiving an M-SEARCH request with the Search Target defined above shall respond as defined in Section 1.3.3 of [1], including a LOCATION header containing an absolute HTTP URL for the UPnP description of the root device. The host portion of the URL SHALL either resolve to an IPv4 address or be an IPv4 address.

The Search Target header (ST) of the response SHALL contain the identifier defined in Section 5.1.

## 5.3 Device description request

On receipt of the M-SEARCH response, the DIAL client shall issue an HTTP GET request to the URL received in the LOCATION header of the M-SEARCH response. (*Note that matching of SSDP header field names is case-insensitive*).

## 5.4 Device description response

On receipt of a valid HTTP GET for the device description, a DIAL server SHALL respond with an HTTP response containing the UPnP device description as defined in Section 2 of [1].

In addition to the requirements of [1], the request SHALL NOT be redirected.

If the request is successful, the HTTP response SHALL contain an additional header field, Application-URL, the value of which SHALL be an absolute HTTP URL. This URL, minus any trailing slash ('/') character, identifies the DIAL REST Service and is referred to as the *DIAL REST Service URL*. The host portion of the URL SHALL either resolve to an IPv4 address or be an IPv4 address.

The format of the Application-URL header field is defined as follows, following the ABNF notation of [2]:

```
Application-URL = "Application-URL" ":" absoluteURI
```

A DIAL client receiving this response SHALL use the provided URL to access the DIAL REST service defined below. (*Note that matching of HTTP header names is case-insensitive*)

### Implementation Note:

- *The UPnP friendlyName field of the device description may be useful for presentation in the DIAL client device UI, for example when offering users a choice of DIAL server devices to interact with.*

## 6 DIAL REST Service

The DIAL REST service represents applications (for example Netflix, YouTube, etc.) as resources identified by URLs. Operations related to an application are performed by issuing HTTP requests against the URL for that application. This URL is known as an *Application Resource URL*.

The Application Resource URL for an application is constructed by concatenating the *DIAL REST Service URL*, a single slash character (`/`) and the *Application Name*.

The Application Name for each application is defined by the application provider. Application Names **MUST** be registered in the DIAL Registry (See Section 7).

### 6.1 Querying for application information

#### 6.1.1 Client request

A DIAL client that wishes to discover information about an application **SHALL** send an HTTP `GET` request to the Application Resource URL.

#### 6.1.2 Server response

On receipt of a `GET` request, the DIAL server **SHALL** first extract the Application Name from the request URL. If the `GET` request is invalid or cannot be processed to extract the Application Name, the server **SHALL** return the appropriate HTTP response code as defined in [2].

If the Application Name is not recognized, the server **SHALL** return an HTTP response with response code `404 Not Found`.

Otherwise the DIAL server **SHALL** return an HTTP response with response code `200 OK`. The MIME type of the response **SHALL** be `text/xml` and the character encoding **SHALL** be UTF-8 and **SHALL** be explicitly indicated with the `charset` MIME parameter.

The XML document SHALL conform to the schema defined in Annex A except that unrecognized XML elements and attributes MUST be ignored by the client.. The semantics of this document are described in the following table:

Element or @attribute	Definition
name	Contains the Application Name
options	
@allowStop	Valid values are: <ul style="list-style-type: none"> <li>• true: Indicates that the DELETE operation described in Section 6.4.2 is supported. This is the default value.</li> <li>• false: Indicates that the DELETE operation described in Section 6.4.2 is not supported.</li> </ul>
state	Valid values are: <ul style="list-style-type: none"> <li>• running: Indicates that the application is installed and either starting or running.</li> <li>• stopped: Indicates that the application is installed and not running.</li> <li>• installable=&lt;URL&gt;: Indicates that the application is not installed but is available for installation. The &lt;URL&gt; in the value MUST contain an absolute HTTP URL. The host portion of the URL SHALL either resolve to an IPv4 address or be an IPv4 address. A DIAL client may initiate installation of the application on the DIAL server by sending an HTTP GET request to the provided URL.</li> </ul> <p>Any other value is invalid and SHOULD be ignored.</p>
link	This optional element SHOULD be included when an application is running <i>and</i> the DIAL server supports DELETE requests as described in section 6.4.2.
@rel	The attribute value MUST be "run" .
@href	The href attribute contains the resource name of the running application, e.g. "run" or "pid-25352". This name SHOULD match the last portion of the name returned in the 201 CREATED response (e.g. http://192.168.1.200:2342/apps/YouTube/run).
additionalData	Contains zero or more elements that the specified application posted, upon being launched, to the DIAL server. The names and values of those elements are outside the scope of this document and are selected by the first-screen application developer.

DIAL servers MAY support client triggering of application installation for specific applications that are not currently installed.

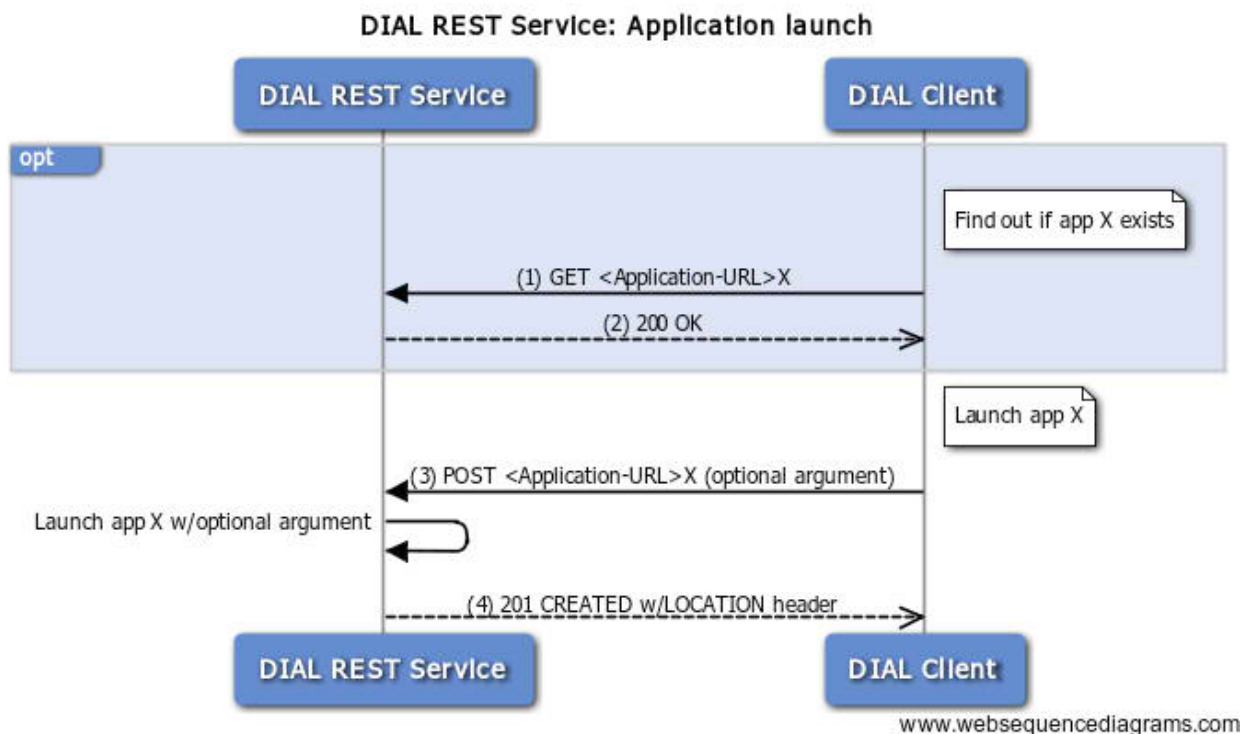
If the Application Name is recognized, the application is not installed, and the DIAL server supports client triggering of application installation for this application, then the DIAL server SHALL return a state element the value of which begins with the string `installable=` as described in the table above.

On receipt of such a request at the DIAL server, if the Application Name is recognized, the application is not installed, and the DIAL server does not support triggering of application installation for this Application Name, then the DIAL server SHALL return an HTTP response with response code 404 Not Found.

*Implementation note:*

- *If an application is installable, the client can choose whether or not to GET the installable URL provided. For example, the client (second-screen application) can present a choice to the user: “Application is not installed. Would you like to install it now?”. The way the first-screen device handles the installation request is OEM-defined; options include immediate installation of the application, presentation of the application in a marketplace or similar application store, etc.*

## 6.2 Launching an application



### 6.2.1 Client request

A DIAL client that wishes to launch an application on a DIAL server SHALL send an HTTP `POST` request to the Application Resource URL for the desired application.

*Implementation Note:*

- *Which applications are available to launch using DIAL is entirely at the discretion of the DIAL server implementer.*

The message body of the `POST` request MAY be empty or MAY contain an argument string to be passed to the application on launch.

If the message body of the `POST` request is empty, a "Content-Length: 0" header SHOULD be sent to avoid receiving a possible "411 Length Required" error.

If the message body of the `POST` request is non-empty, the MIME type SHALL be `text/plain`, the character encoding SHALL be UTF-8, and the character encoding SHALL be indicated explicitly by including the `charset` MIME parameter.

*Implementation Notes:*

- *How the argument is passed to the application may be platform- and application-specific and is outside the scope of this document.*
- *The format of the argument should match the requirements of the application being launched. Key-value pairs, JSON, XML, etc. are all possible and the choice is outside the scope of this document.*
- *DIAL servers MUST support argument strings up to 4KB in length. The DIAL server implementer MAY choose to support larger argument strings.*
- *A DIAL server MAY pass the argument string directly to the application: Applications MUST NOT assume that any security checks (such as character-encoding checks) have been performed and therefore MUST perform their own security checks on the argument data.*
- *The DIAL payload MUST be passed to the launched application in a way that ensures it **cannot be used to circumvent fundamental security or integrity of the platform**. For example, the DIAL payload SHOULD NOT be used to*
  - *directly define arbitrary command-line parameters when running a native application unless adequate protections are taken.*
  - *provide the actual executable name.*
  - *control process attributes like priority or ownership.*

### 6.2.2 Server response

On receipt of a valid `POST` request, the DIAL server SHALL first extract the Application Name from the `POST` URL. If the `POST` request is invalid or cannot be processed to extract the Application Name, the server SHALL return the appropriate HTTP response code as defined in [2].

The server SHALL respond, as indicated in the table below, depending on:

- whether the Application Name is not recognized.
- whether the content length of the message body exceeds the maximum size supported by the server.
- the current application state:
  - not started: the application is not running.
  - starting: the application is being started due to another DIAL REST Service request or for another reason.
  - running: the application is running.
- whether the message body is empty or non-empty.

DIAL servers MUST support any message body in a POST request with content length of 4KB or less.

In the table below, earlier rows take precedence over later ones. If the Action indicates a HTTP response code the server SHALL return an HTTP response with the indicated response code.

Application recognized	Message body	Application state	Action
No	any	n/a	404 Not Found
Yes	too long	n/a	413 Request Entity Too Large
Yes	empty	Not running	201 Created * Start application
Yes	non-empty	Not running	201 Created * Start application with provided argument, if any
Yes	empty	Starting <sup>1</sup>	200 OK
Yes	non-empty	Starting <sup>2</sup>	200 OK
Yes	empty	Running	200 OK
Yes	non-empty	Running	Provide new argument to application

<sup>1</sup> Started by DIAL or by any other means, e.g. built-in menu, etc.

<sup>2</sup> Started by DIAL or by any other means, e.g. built-in menu, etc.

If the application is running after the action specified above, the DIAL server SHALL return an HTTP response with response code 201 `Created`. In this case, the `LOCATION` header of the response shall contain an absolute HTTP URL identifying the running instance of the application, known as the *Application Instance URL*. The host portion of the URL SHALL either resolve to an IPv4 address or be an IPv4 address. No response body shall be returned.

Otherwise, if the application cannot be successfully started or re-started for any reason, the DIAL server SHALL return an HTTP response with response code 503 `Service Unavailable`.

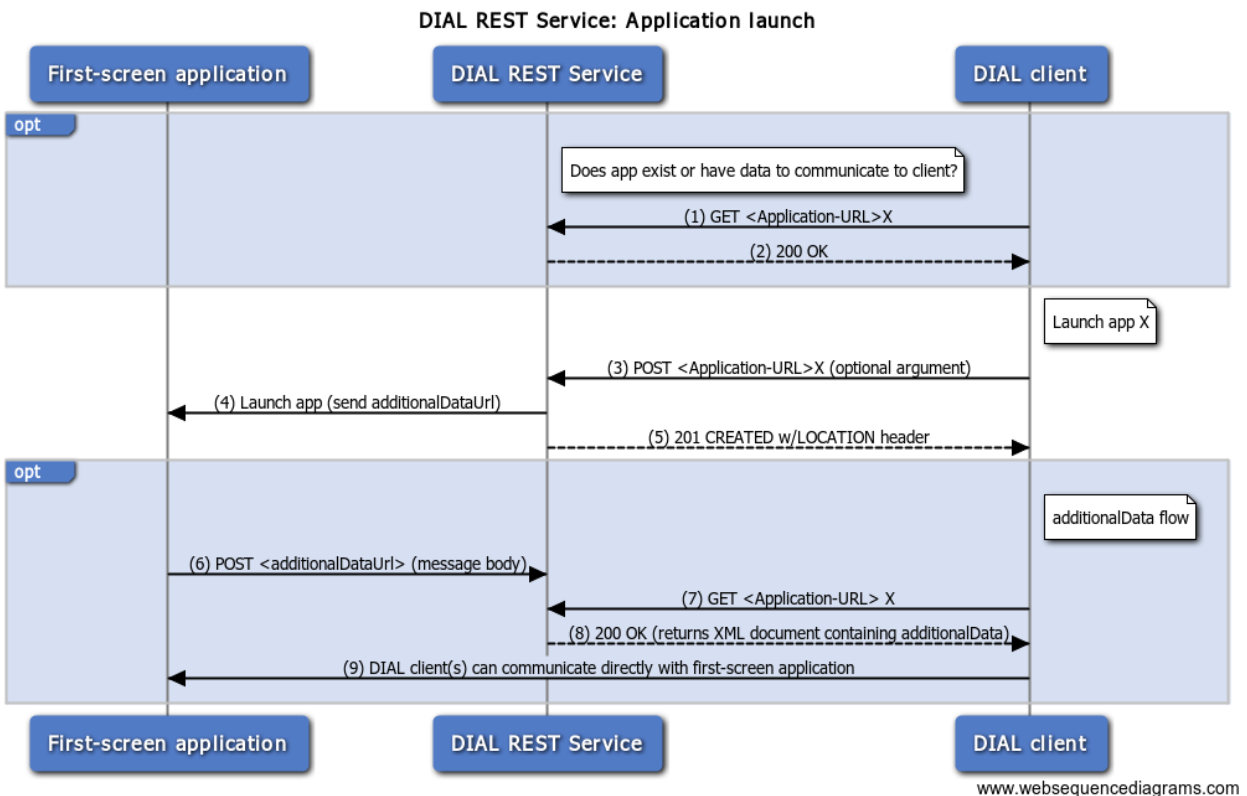
*Implementation Note:*

- *If the application is already running and an argument is provided and the platform does not support providing new arguments to running applications, then the application should not be restarted with the new argument unless allowed by the application provider.*
- *To run the application, the host system should use a non-blocking form of "fork", "exec", "spawn", or equivalent that will catch any immediate error (like `ENOMEM` or `EAGAIN`) and map it to a 503 "Service Unavailable" error. If there is no error and the system command to run the application succeeds, then the application is running for the purposes of this specification.*

The Application Instance URL may be used to stop the running instance of the application, as described in section 6.4.

### 6.3 Sending additional data from a first-screen application to a DIAL server

An application launched or running on the first-screen device can send a small amount of data to the DIAL server. The DIAL server can then communicate that information to DIAL clients that connect to the DIAL server. The additional data could, for example, permit multiple second-screen devices to simultaneously connect to and interact with the first-screen application.



The process diagram above illustrates the process by which a first-screen application can send data back to the DIAL server. The DIAL server can subsequently communicate to any second-screen device that contacts it. The following notes explain the image in more detail:

- Steps 1 through 5 show the same process as the image in section 6.2.
- Steps 1 and 7 represent the same type of request.
- Steps 2 and 8 represent the same type of response. Step 8 illustrates a situation in which the DIAL server's response contains an `<additionalData>` element.
- A DIAL server must support CORS to support the request made in step 6.
- Any process or mechanism that launches a first-screen application (step 4), such as a menu on the first-screen device, **MUST** send the `additionalDataUrl` shown in step 4 when launching the application. The `additionalDataUrl` enables steps 6 through 9 to occur regardless of which process launches the first-screen application.
- A second-screen application running on a DIAL client **MAY** be designed to check the



<state> element in the response sent in steps 2 and 8 and follow a different flow if the application is already running. A second-screen application that does follow a different process depending on an application's state MUST implement steps 1 and 2 even though they are marked as optional.

For example, if a first-screen application supports multiple second-screen clients simultaneously, then a second-screen client must send the request in step 1 to determine whether the first-screen application is running. If the application is not running, the second-screen client launches the application (step 3). However, if the application is running, the second-screen client follows a different process (step 9)

### 6.3.1 Request to POST additional data from first-screen application to DIAL server

When a first-screen application is launched, the launching service (DIAL server, TV menu, etc.) can send an `additionalDataUrl` to the application. The `additionalDataUrl` can be sent as a parameter, property, key, etc., which is a decision that is left to the DIAL server implementer.

If present, that parameter SHALL specify a URL that the first-screen application can use to POST additional data, structured as key-value pairs, to the DIAL server. (The DIAL server can subsequently relay that information to other DIAL clients that connect to the server as discussed in section 6.1.3.)

- If the `additionalDataUrl` is specified, that value MUST be sent to the first-screen application every time it is launched, even when the first-screen application is not started by the DIAL server. This ensures that the first-screen application can always communicate the additional data to the DIAL server.
- The URL MUST be unique for each application.
- The `additionalDataUrl` value MUST be URL-encoded using the rules defined in [5] for MIME type `application/x-www-form-urlencoded`.
- The URL host MUST be `localhost`. This value SHALL be checked in the application.
- Using an `additionalDataUrl` of the form `/apps/<app_name>/dial_data` is RECOMMENDED. If the URL contains the application name, the DIAL server can easily identify the application to which data is being sent.

If a first-screen application receives an `additionalDataUrl`, the application should POST a request to that URL as soon as the additional data is available. The size of the POST request MUST be smaller than 4KB. The key names and their corresponding values are selected by the application provider and are outside the scope of this document.

If present, that parameter SHALL specify a URL that the first-screen application can use to POST additional data, structured as key-value pairs, to the DIAL server.

#### *Security considerations:*

A first-screen application running on the application provider's domain needs to be able to call the `additionalDataUrl` that the DIAL server provides. To support this capability, the DIAL server MUST support Cross-Origin Resource Sharing (CORS) requests as defined in [4].

For example, the YouTube TV application running on [www.youtube.com](http://www.youtube.com) needs to call the DIAL server's `additionalDataUrl`. To provide this support, the DIAL server needs to add the following response header when handling requests to the application's `additionalDataUrl` (`/apps/youtube/dial_data`):

```
Access-Control-Allow-Origin: https://www.youtube.com
```

*Implementation Note:*

- *Each application MUST have a way to communicate its authorized domain to the DIAL server. This communication could occur at build time or by updating stored metadata. A hardcoded list of authorized domains MAY only be used for systems that cannot download applications dynamically.*

### 6.3.2 Process for handling `additionalData` on a DIAL server

The DIAL server MUST store the key-value pairs sent in the body of a POST request from a first-screen application to the `additionalDataUrl`. The DIAL server MUST then include those key-value pairs, formatted as XML elements and element values, in responses (section 6.1.2) to all subsequent DIAL Application Information requests (section 6.1.1) pertaining to the same application.

Each key-value pair is returned as a separate XML element nested inside the `additionalData` element as defined below:

- The element name is the same as the key in the key-value pair.
- The element value is set to `xmlencode(urldecode(value))`

Each time a DIAL server receives a POST request sent to an `additionalDataUrl`, the server MUST replace all stored key-value pairs for the sending application with the key-value pairs in the request. For example, an application could clear a value by sending a request that omits a key-value pair sent in a previous request.

The key-value pairs stored for an application MUST always be returned in a DIAL Application Information response, even if the application is not currently running.

## 6.4 Stopping an application

### 6.4.1 Client request

A DIAL client that wishes to stop a running instance of an application on a DIAL server SHALL send an HTTP `DELETE` request to the Application Instance URL.

### 6.4.2 Server response

Support of the `DELETE` request is RECOMMENDED for DIAL servers.

If the `DELETE` request is not supported the DIAL server SHALL return an HTTP response with response code `501 NOT IMPLEMENTED`.

If supported, then on receipt of a `DELETE` request, the DIAL server SHALL first determine whether the provided URL corresponds to a running application. If the `DELETE` request is invalid or cannot be processed to determine whether the provided URL corresponds to a running application, the server SHALL return the appropriate HTTP response code as defined in [2].

If the provided URL does not correspond to an application running in the foreground, the server SHALL return an HTTP response with response code `404 Not Found`.

Otherwise, the server SHALL send an HTTP response with response code `200 OK` and attempt to stop the running application. These two operations SHOULD be carried out asynchronously.

## 7 DIAL Registry

To ensure that the correct name for each application is well-defined and to avoid naming conflicts, Application Names must be registered in the DIAL Registry. The DIAL specification maintainer will maintain this registry and make it available to anyone implementing or using the DIAL protocol.

Application Names may be registered explicitly, or a set of Application Names with a common prefix may be registered by registering an *Application Prefix*.

Application Names and Prefixes MUST consist of a sequence of characters matching the `pchar` production of RFC3986 [3]. Matching of Application Names and Prefixes is case-sensitive and SHALL be performed after decoding percent-encoded characters (i.e. the three-character sequence `'%30'` matches the single character `'0'`.)

An Application Prefix MUST be at least four (4) characters in length (after decoding percent-encoded characters) and must include a recognizable company name. For example: `"Acme-"`, `"com.acme"`.

Application names can be registered if the application is actually available in the market and there is no conflict with previously registered names or prefixes. Application prefixes can be registered if the company name in the prefix is actively delivering applications or devices that run them to the market. Names or prefixes that may be confused with previous registrations (e.g. `"netflix"`, `"youtube"`, `"Y0uTube"`, etc.), or are not intended for use with DIAL, may not be registered.

## 8 References

- [1] UPnP™ Device Architecture 1.1 , 15 October 2008,  
<http://upnp.org/sdcp-s-and-certification/standards/device-architecture-documents/>
- [2] RFC2616 (<http://www.ietf.org/rfc/rfc2616.txt>)
- [3] RFC3986 (<http://tools.ietf.org/html/rfc3986>)
- [4] Cross-Origin Resource Sharing - W3C Recommendation, <http://www.w3.org/TR/cors/>
- [5] HTML 4.01 Specification (<http://www.w3.org/TR/html401/>)

## 9 Acknowledgements

This document greatly benefited from review, feedback, and suggestions from several CE manufacturers and content providers. The document authors appreciate their help and support.

Notably, Samsung and Sony provided significant guidance to ensure that DIAL would be a compatible and effective solution for first-screen devices and also meet their goals for great second-screen user experiences. The document authors especially thank our colleagues at these companies for their efforts on behalf of DIAL.

## Annex A: Application resource XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:dial-multiscreen-org:schemas:dial"
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="urn:dial-multiscreen-org:schemas:dial">
  <xs:element name="service" type="ServiceType"/>

  <xs:complexType name="ServiceType">
    <xs:sequence>
      <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="options" type="optionsType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="state" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="link" type="linkType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="additionalData" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:any minOccurs="0" processContents="lax"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="dialVer" type="xs:string" use="optional"/>
  </xs:complexType>

  <xs:complexType name="optionsType">
    <xs:attribute name="allowStop" type="xs:boolean" use="optional"/>
  </xs:complexType>

  <xs:complexType name="linkType" mixed="true">
    <xs:attribute name="href" use="required" type="xs:anyURI"/>
    <xs:attribute name="rel" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:schema>
```

## Annex B: Message Examples

### B.1 M-SEARCH

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: seconds to delay response
ST: urn:dial-multiscreen-org:service:dial:1
USER-AGENT: OS/version product/version
```

### B.2 M-SEARCH response

```
HTTP/1.1 200 OK
LOCATION: http://192.168.1.1:52235/dd.xml
CACHE-CONTROL: max-age=1800
EXT:
BOOTID.UPNP.ORG: 1
SERVER: OS/version UPnP/1.1 product/version
ST: urn:dial-multiscreen-org:service:dial:1
```

### B.3 Device description request

This message is sent from the DIAL client to the DIAL server:

```
GET /dd.xml HTTP/1.1
...
```

### B.4 Device description response

```
HTTP/1.1 200 OK
Application-URL: http://192.168.1.1:12345/apps
...
<UPnP device description in message body>
```

### B.5 Application information request

This message is sent from the DIAL client to the DIAL server:

```
GET /apps/YouTube
...
```



## B.6 Application information response

```
HTTP/1.1 200 OK
...
<?xml version="1.0" encoding="UTF-8"?>
<service xmlns="urn:dial-multiscreen-org:schemas:dial" dialVer="1.7">
  <name>YouTube</name>
  <options allowStop="true"/>
  <state>running</state>
  <link rel="run" href="http://192.168.1.1:12345/apps/YouTube/run"/>
</service>
```

## B.7 Application information response for application that allows multiple connected users

```
HTTP/1.1 200 OK
...
<?xml version="1.0" encoding="UTF-8"?>
<service xmlns="urn:dial-multiscreen-org:schemas:dial" dialVer="1.7">
  <name>YouTube</name>
  <options allowStop="true"/>
  <state>running</state>
  <link rel="run" href="http://192.168.1.1:12345/apps/YouTube/run"/>
  <additionalData>
    <screenId>screen123</screenId>
    <sessionId>token123</sessionId>
  </additionalData>
</service>
```

Note: The `<screenId>` element is provided as an example. In fact, the `<additionalData>` element can contain one or more elements. The names and values of those elements are outside the scope of this document and are selected by the first-screen application developer.

## B.8 Application launch request

This message is sent from the DIAL client to the DIAL server:

```
POST /apps/YouTube
Content-Type: text/plain; charset="utf-8"
...
param1=value1&param2=value2
```

## B.9 Application launch request with additionalDataUrl

This message is sent from the DIAL client to the DIAL server:

```
POST /apps/YouTube
Content-Type: text/plain; charset="utf-8"
...
param1=value1&param2=value2
```

The DIAL server launches the first-screen application and passes the `additionalDataUrl` argument to that application in an implementation-specific manner.

For browser-based applications, the `additionalDataUrl` should be appended to the launch URL as an additional query-string argument separate from the DIAL payload.

For example, the URL used to launch the first-screen application might look like this:

```
https://example.com/tv?dialpayload=<URL-Encoded DIAL
payload>&additionalDataUrl=<URL-Encoded URL>
```

## B.10 Application launch response

```
HTTP/1.1 201 CREATED
LOCATION: http://192.168.1.1:12345/apps/YouTube/run
...
```

## B.11 Application request sending additional data to DIAL server

```
POST /apps/YouTube/dial_data
Host: localhost:8080
Content-Type: text/plain; charset="utf-8"
screenId=screen123&sessionId=token123
```

## Annex C: Sequence diagram source

```
title DIAL Discovery
participant "UPnP server" as U
participant "DIAL server" as S
participant "DIAL client" as C
note right of C: Discover DIAL enabled TV
C->>S: (1) M-SEARCH
S-->>C: (2) response with LOCATION header
note right of C: Look up DIAL REST service
C->>U: (3) HTTP GET LOCATION URL
U-->>C: (4) UPnP Device Description with Application-URL header

title DIAL REST Service: Application launch
participant "DIAL REST Service" as S
participant "DIAL client" as C
opt
note right of C: Find out if app X exists
C->>S: (1) GET <Application-URL>X
S-->>C: (2) 200 OK
end
note right of C: Launch app X
C->>S: (3) POST <Application-URL>X (optional argument)
S->>S: Launch app X w/optional argument
S-->>C: (4) 201 CREATED w/LOCATION header

title DIAL REST Service: Application launch
participant "First-screen application" as F
participant "DIAL REST Service" as S
participant "DIAL client" as C
opt
note right of S: Does app exist or have data to communicate to client?
C->>S: (1) GET <Application-URL>X
S-->>C: (2) 200 OK
end
note right of C: Launch app X
C->>S: (3) POST <Application-URL>X (optional argument)
S->>F: (4) Launch app (send additionalDataUrl)
S-->>C: (5) 201 CREATED w/LOCATION header
opt
note right of C: additionalData flow
F->>S: (6) POST <additionalDataUrl> (message body)
C->>S: (7) GET <Application-URL> X
S-->>C: (8) 200 OK (returns XML document containing additionalData)
C->>F: (9) DIAL client(s) can communicate directly with first-screen application
end
```

## CHANGE HISTORY

### Version 1.7.1

1. Modified Annex B.9 to clarify the use of `additionalDataUrl`.
2. Modified Annex B.3, B.5, B.8, B.9 to clarify what sends and receives the message examples.

### Version 1.7

1. Modified the protocol to explain how a first-screen application can send data back to the DIAL server, which the DIAL server can communicate to any second-screen device that contacts it.
  - a. Added steps to sections 3.1, 3.2, and 3.3.
  - b. Added section 3.4, which explains use case of a second-screen application connecting to an already running first-screen application.
  - c. 6.1.2: Added `<additionalData>` element to table.
  - d. 6.3.1: Added instructions for including `additionalDataUrl` in request to launch first-screen application. Note that the `additionalDataUrl` **MUST** be sent to the first-screen application every time it is launched, even when the first-screen application is not started by the DIAL server.
  - e. 6.3.2: Added section explaining how to send additional data from a first-screen application to a DIAL server and how to relay that information from the DIAL server to DIAL clients that try to launch or retrieve information about that application.
  - f. Annex A:
    - i. Modified XML schema to contain `<additionalData>` element.
    - ii. Modified XML schema to add an optional DIAL version attribute to the `<service>` element of the Application Information response.
  - g. Annex B: Reordered sample messages and added messages:
    - i. B.7 (Application Information response for application that allows multiple connected users)
    - ii. B.9 (Application launch request with `additionalDataUrl`)
    - iii. B.11 (Application request sending additional data to DIAL server)
2. Reordered subsections of 6.1 as follows:
  - 6.1: Querying for application information
  - 6.2: Launching an application
  - 6.3: Sending additional data from a first-screen application to a DIAL server
  - 6.4: Stopping an application
3. 6.1.2: Cleaned up element and attribute definitions. Moved details about installable applications to the description of the `installable=` value in the `<state>` element's definition.



## Version 1.6.5

1. Modified section to 6.1.3.2 to break down the `rel` and `href` attributes in separate rows. Updated the schema in Annex A to no longer reference the Atom schema's definition of `LinkType`. `LinkType` is now entirely defined in this spec.
2. In section 6.1.1.2 clarified that the Application Instance URL may only be used to **stop** the running instance of the application and not to request information.
3. Replaced 'backslash' with 'slash' in paragraphs 5.4 and 6.1 to describe the '/' character in a URL.
4. Specified that the default value for the `@allowStop` attribute is true in section 6.1.3.2

## Version 1.6.4

5. Title: Cleaned up title page.
6. 6.1.1.1: Added recommendation to send a "Content-Length: 0" header when POST is empty to avoid receiving a possible "411 Length Required" error.

## Version 1.6.3

1. 6.1: Simplified the explanations of the concatenation rules for the application name with Application-URL.
2. 6.1.1.1: Added emphasis on security measures when handing off DIAL payload to application in implementation notes.
3. 6.1.1.2: Clarified we are talking about the LOCATION header.
4. 6.1.1.2: Implementation notes: Clarified that restarting an application due to a modified DIAL payload is only allowed if an application provider allows it.
5. 6.1.3.2: Clarified that an application that has just launched (is starting) should be reported as "running".
6. Added CHANGE HISTORY section to document.