

The RIAP protocol

Background.

The riap protocol came to life as a solution for the [issue 374 "Add support for pseudo protocols"](#)¹. Which in turn started of based from discussion on [issue 371 "Optimize Internal Calls"](#)².

About riap.

The riap://... scheme identifies a so-called pseudo-protocol (terminology derived from Apache Cocoon pointing out the difference between 'real' or 'official'/ public protocols and this scheme which only relates to internal processing of the restlet system architecture.

RIAP is short for 'Restlet Internal Access Protocol'.

In short, it is just a mechanism that allows various applications to call upon each other to retrieve resources for further processing.

NOTE

"Application" here refers to the strict restlet API notion. Same remark should be made on the usage of "Component" further down.

The riap:// scheme brings a URI-notation for these inter-application calls. With that URI-notation those calls fall naturally under appliance of the uniform interface.

There are two blends of this riap protocol, that use distinct 'authorities' for different use cases.

- riap://component/**
 - resolves the remainder-URI with respect to the current context's "component"
- riap://application/**
 - resolves the remainder-URI with respect to the current context's "application" (so applications could use this scheme to call resources within themselves!)

The riap use case.

Think about some application X that needs a resource Y that is available on a configurable base-uri. Whatever that baseUri is, the application-implementation should just use the `context.getClientDispatcher()` to `get()` that needed resource.

So basically the dependency from application X to "whatever service that will offer me resource Y" can be expressed in this base-URI. Thinking about dependency injection solutions, the configuration of X is covered by calling some `setResourceYBaseUri(...)`. That will cover any of these 'service-providers' of this resource Y:

- file://whateverpath/Y
 - a local static file
- http://service.example.com/somepath/Y
 - an external service provider

1. http://restlet.tigris.org/issues/show_bug.cgi?id=374
2. http://restlet.tigris.org/issues/show_bug.cgi?id=157

- riap://component/internalpath/Y
 - a URI resolved versus the internal router of the current 'Component'
- riap://application/fallback/Y
 - this application itself providing some basic version of the required resource.

Next to this expression flexibility of dependencies, this riap:// approach ensures some extra

- efficiency:
 1. It avoids going through the actual network layers versus the alternative of calling http://localhost/publicpath/Y
 2. It offers direct access to the Response and Representation objects allowing to bypass possibly useless serialization-deserialization (e.g. for sax and or object representations)
- shielding:
 1. It offers the possibility to keep the internal component offering resource Y to be available only internally (not publicly over http://hostname/publicpath/Y)

Bottom line: the basic idea behind this riap:// is one of flexibly decomposing your 'component' in smaller reusable/configurable/interchange-able 'applications' while assuring optimal efficiency when calling upon each other.

How to use.

Calling the riap:

To call a resource via the riap:// scheme one can just use:

```
Context context;

context.getClientDispatcher().get("riap://component/some-path/Y");
```

There is no need to register a RIAP client, this is handled by built-in support.

Making resources available to the riap

Applications need just to be attached to the internalRouter property of the Component.

```
Component component;

component.getInternalRouter().attach("/path", someApplication);
```

NOTE

Applications can be attached multiple times (at different paths) to both the internal-router as to several virtual hosts.

A 'pure internal' application should only be attached to the internal router.

Caution

Note that internal/dynamic resources that require protocol specific attributes of the URI where it is invoked (like hostname) might yield errors or unexpected results when called via the riap internal protocol.

Also beware that constructed URI's to secondary resources which are based on the 'own reference' can yield unexpected or at least different results when the resource is suddenly being called via the riap.

Give these cases some special attention during your design, specially when porting stuff from other environments (e.g. servlet app)

Feature Availability

This feature landed in svn-trunk end of November 2007 (rev 2251). It will be part of the 1.1 release.

FIXME

TODO: check that last statement.