<grant-uri-permission>

SYNTAX:

```
<grant-uri-permission android:path="string"
                      android:pathPattern="string"
                      android:pathPrefix="string" />
```

CONTAINED IN:

`<provider>`

DESCRIPTION:

Specifies which data subsets of the parent content provider permission can be granted for. Data subsets are indicated by the path part of a `content:` URI. (The authority part of the URI identifies the content provider.) Granting permission is a way of enabling clients of the provider that don't normally have permission to access its data to overcome that restriction on a one-time basis.

If a content provider's `grantUriPermissions` attribute is "`true`", permission can be granted for any the data under the provider's purview. However, if that attribute is "`false`", permission can be granted only to data subsets that are specified by this element. A provider can contain any number of `<grant-uri-permission>` elements. Each one can specify only one path (only one of the three possible attributes).

For information on how permission is granted, see the `<intent-filter>` element's `grantUriPermissions` attribute.

ATTRIBUTES:

`android:path`
`android:pathPrefix`
`android:pathPattern`

A path identifying the data subset or subsets that permission can be granted for. The `path` attribute specifies a complete path; permission can be granted only to the particular data subset identified by that path. The `pathPrefix` attribute specifies the initial part of a path; permission can be granted to all data subsets with paths that share that initial part. The `pathPattern` attribute specifies a complete path, but one that can contain the following wildcards:

- An asterisk ('`*`') matches a sequence of 0 to many occurrences of the immediately preceding character.
- A period followed by an asterisk ("`.*`") matches any sequence of 0 to many characters.

Because '`\`' is used as an escape character when the string is read from XML (before it is parsed as a pattern), you will need to double-escape: For example, a literal '`*`' would be written as "`\\*`" and a literal '`\`' would be written as "`\\\\`". This is basically the same as what you would need to write if constructing the string in Java code.

For more information on these types of patterns, see the descriptions of `PATTERN_LITERAL`, `PATTERN_PREFIX`, and `PATTERN_SIMPLE_GLOB` in the `PatternMatcher` class.

INTRODUCED IN:

API Level 1

SEE ALSO:

the `grantUriPermissions` attribute of the `<provider>` element