

This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. The article is believed to be still valid for the original topic scope.



## Contents

- [1 Fetcher](#)
- [2 Pollable services](#)
- [3 Notifications](#)
  - ◆ [3.1 New content notification](#)
    - ◇ [3.1.1 Arguments](#)
  - ◆ [3.2 Fetcher error notification](#)
    - ◇ [3.2.1 Arguments](#)
- [4 Content caching](#)
- [5 Poll period](#)
- [6 Aggregation](#)
- [7 See also](#)

## Fetcher

The Fetcher is a generic server side component that provides automatic polling and content caching functionality for services used in widgets.

Polling means that the Fetcher will automatically and periodically perform a *fetch*-action on the target *fetchable* defined by the parameters of the service. The *fetch*-action returns *content* that the Fetcher caches to a memory-resident cache. By comparing the previously *fetches content* with the newly *fetches content*, the Fetcher detects if any new content is available and notifies the listening clients accordingly.

*Fetch* action, *fetchable*, and *content* are abstract concepts, but typically *fetching* means getting content from an external service, like fetching the content of an RSS feed.

## Pollable services

The Fetcher works with services that support polling. Currently the syndication and webfeed services support polling.

## Fetcher\_details

A service must be marked as *pollable* in order to be polled. A service is marked as pollable by defining a service parameter `polltype` in the widget's *widget.xml* file. The value of the `polltype` service parameter tells if the service should be polled and how it should be polled. Currently, the only valid value for the parameter is `auto`.

For example, a syndication service is defined as pollable as follows:

```
...
<service type="syndication" id="feed">
  ...
  <reference from="polltypeparam" to="polltype"/>
  ...
</service>
...
<parameters>
  ...
  <parameter type="string"
    name="polltypeparam"
    description="Poll type"
    help="To poll or not and how"
    editable="false">auto</parameter>
  ...
</parameters>
...
```

The Fetcher detects pollable services and initiates polling automatically when

- the client connects,
- when new widgets appear on dashboard, or
- when widget's service parameters change.

Polling is kept alive automatically while the client is connected.

## Notifications

The Fetcher notifies the listening clients about new content and possible error conditions. The Fetcher does not push the actual content to the clients, but the clients should react to the new content notifications and use the `dabo/call` service with the pollable service type specific actions and arguments to get the content.

The notification are sent sent to the client by using the `dabo/push/notify` service. Currently there are two notifications:

- new content notification and
- fetcher error notification.

## New content notification

The new content notification is sent to listening clients when the Fetcher performs a poll and finds new content items in the fetched content.

## Fetcher\_details

The syntax of the new content notification is:

```
input      = (list (int widgetid) (string serviceid) (use NewContent))

NewContent = (list (bag (required (bind (const type)      (const newcontent)))
                    (optional (bind (const numnew)      (int newCount)))
                    (optional (bind (const newesttts)    (int newestTimestamp)))))
```

### Arguments

Name	Type	Description
<b>type</b>	const	Tag value <code>newcontent</code> identifying the notification type.
<code>numnew</code>	int	Tells the number of new content items found since the previous poll.
<code>newesttts</code>	int	The create time of the newest content item. The timestamp value is in milliseconds since the 1st of January, 1970, and interpreted as UTF/GMT timestamp.

## Fetcher error notification

The fetcher error notification is sent to the client if an error occurs while initiating or performing polling, or when a serious error occurs while executing client-initiated content action request on a pollable service.

The syntax of a fetcher error notification is:

```
input      = (list (int widgetid) (string serviceid) (use FetcherError))

FetcherError = (list (bag (required (bind (const type)      (const fetchererr)))
                    (required (bind (const errkey)      (string  errorKey)))
                    (optional (bind (const errmsg)      (string  errorMsg)))
                    (optional (bind (const pollstop)    (boolean isPollingStopped)))))
```

### Arguments

Name	Type	Description
<b>type</b>	const	Tag value <code>fetchererr</code> identifying the notification type.
<b>errkey</b>	string	Contains the error key or code. Most error keys are specific to the pollable service, but there are also some generic ones. The syntax of the error key value is <code>context.key</code> . The <code>context</code> part is the service type (like <code>syndication</code> ) for service specific errors and <code>fetcher</code> for generic errors. The <code>key</code> part is the actual error key.
<code>errmsg</code>	string	Contains the error message. This message is not localized and is meant for admins and developers only, not for end users.
<code>pollstop</code>	boolean	Tells if the error caused the polling to stop for the pollable service. Some error conditions such as errors in the widget configuration cause polling

	<p>to stop immediately, because they can be corrected only by editing the widget configuration. Some error conditions such as errors in the external service provider's service, or in returned content, do not stop polling immediately and the Fetcher keeps trying a couple of times. If error condition persists for a couple of polls, then polling is stopped.</p>
--	--

## Content caching

When the client requests for content, the content is returned from the Fetcher's memory-resident cache if already there. Requesting for content does then not cause requests to external services unless necessary or specifically requested.

The Fetcher's memory-resident cache does not store any old content, only the current data received in the latest poll.

A client can force cache refresh and new content fetch from external service by using the `refresh => true` action argument when getting content with `dabo/call`. Currently this works with the `getItems` action of the syndication and webfeed services.

Note that content caching is done for a service that supports polling even if the service instance is not marked as pollable with the `polltype` service parameter. If, for example, a syndication service is not polled and the widget needs fresh content, the `refresh => true` action argument should then be used with the `getItems` action.

## Poll period

By default, the Fetcher constantly adjusts the poll period of a pollable service. The Fetcher recalculates the poll period after each poll. The algorithm basically calculates mean over the periods between content items (whatever "content item" means for the pollable service type -- for syndication service it means "feed item"), weighting newest items more than older ones. If a poll returns only one new item, then the poll period is optimal and is not changed for the next poll.

The Fetcher does not set the poll period arbitrarily small or large, but keeps the poll period between minimum and maximum poll period values. The minimum and maximum poll period values have system level defaults. The default values are "20s" and "3h".

The system-level minimum and maximum poll periods can be overridden and set individually for each service instance. The values can be overridden by defining the service parameters `minpollperiod` and `maxpollperiod` in the *widget.xml file*. The values cannot be smaller than 10s.

If a value is a plain number, it is interpreted as a millisecond value. A more human readable way is to use the unit symbols `d` (days), `h` (hours), `m` (minutes), `s` (seconds), and `u` (millisecs) after integer values.

If the `minpollperiod` and `maxpollperiod` are set to the same value, then the Fetcher does not (need to) calculate poll period but will poll at regular intervals.

## Fetcher\_details

Below is an example how to set the minimum and maximum poll periods for a syndication service:

```
...
<service type="syndication" id="feed">
...
  <reference from="minpollperiodparam" to="minpollperiod"/>
  <reference from="maxpollperiodparam" to="maxpollperiod"/>
...
</service>
...
<parameters>
...
  <parameter type="string"
    name="minpollperiodparam"
    description="Min poll period"
    help="Min poll period"
    editable="false">5m</parameter>
  <parameter type="string"
    name="maxpollperiodparam"
    description="Max poll period"
    help="Max poll period"
    editable="false">1h30m</parameter>
...
</parameters>
...
```

## Aggregation

The Fetcher optimizes polling and content access by using aggregation. Aggregation means that similar service instances, that is, all service instances in different clients (dashboards) and widgets having equivalent service parameter values, are served by a single handling task.

The handling task performs polling and content caching, and handles the content access requests from clients. The Fetcher performs the necessary routing and aggregation automatically. The Fetcher handles also distribution of notifications originating from a handling task to all listening clients.

## See also

- [Getting content with Services](#)
- [Available content fetching services](#)
  - ◆ [Syndication service](#)
  - ◆ [Webfeed service](#)
  - ◆ [HTTP service](#)
- **Fetcher details**
  - ◆ [Feed formats](#)
  - ◆ [HTTP authentication](#)
- [Advanced filters](#)
  - ◆ [Filter expressions reference](#)