

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 The service](#)
 - ◆ [1.1 Output fields and field filters](#)
 - ◆ [1.2 Main filter](#)
 - ◆ [1.3 Overriding default field filters](#)
 - ◆ [1.4 Overriding item ID filter](#)
 - ◆ [1.5 Paged fields](#)
 - ◇ [1.5.1 Example](#)
 - ◆ [1.6 Specifying custom field filters](#)
 - ◆ [1.7 Advanced: overriding main filter](#)
- [2 Service Parameters](#)
- [3 Service actions](#)
 - ◆ [3.1 Definitions](#)
 - ◆ [3.2 Action `getItems`](#)
 - ◇ [3.2.1 Arguments](#)
 - ◆ [3.3 Action `getPage`](#)
 - ◇ [3.3.1 Arguments](#)
 - ◆ [3.4 Action `bookmark`](#)
 - ◇ [3.4.1 Arguments](#)
 - ◆ [3.5 Action `senditem`](#)
 - ◇ [3.5.1 Arguments](#)
- [4 Example](#)
- [5 See also](#)

The service

The *webfeed* service can be used to deliver [RSS](#) and [ATOM](#) content to the client. The difference to the [syndication service](#) is that the webfeed service is very customizable. The widget developer can define how the feed content from the service provider is parsed, and what content and in what format is sent to the client. The webfeed service parses the source feed with [advanced filters](#). The service is customized by defining these parsing filters in the *widget.xml* file of the widget.

Webfeed_service

The webfeed service can be used to handle also other than RSS and ATOM feeds provided that the structure of the feed, or at least the structure of the output sent to client, resembles RSS and ATOM feeds. The feed should consist of a list of feed items.

Note. Even though possible, currently this kind of usage is not very intuitive because it requires "creative misuse" of some webfeed service features. There are plans to generalize the webfeed service so that parsers for other feed formats could be defined in a cleaner way.

Feed parsing is not handled with a single filter but a set of filters. One of the filters is the *main filter* and the rest of the filters are the *field filters*. The main filter takes the feed-source as input and extracts the item constructs from the feed one by one. The main filter loops through the parsed item construct and gives each item construct as input to each field filter. Each field filter extracts a specific field data, such as title or author, from the item construct. The main filter gathers the results from the field filters and usually forms a list of items that further consist of fields.

The webfeed service has default parsing filters that provide more or less the same parsing functionality and output as the [syndication service](#). The webfeed service becomes useful if the feed contains non-standard constructs or field data in a format that the syndication service cannot handle in a desired way. Because there is a separate filter for each field, only the filter that parses the field that needs special handling needs to be overridden.

The webfeed service can also be used to restrict the set of fields to be delivered to the client and, more importantly, expand the set of fields with special fields. For example, the RSS feed from www.digg.com contains Digg-specific fields such as "digg counts" and links to Digg users' avatar images. These can be easily parsed with simple filters, one filter for a custom field.

Usually, a customized webfeed widget requires a custom client side [WidSets Scripting Language](#) script.

The main filter can also be overridden. This is the key to parsing other than RSS and ATOM feeds.

Output fields and field filters

The webfeed service assumes that the service output sent to the client consists of a list of items where each item consists of a set of named fields, or more precisely that each item gets constructed by calling a set of field filters.

The desired set of item fields or the desired set of field filters to be called is specified with the service parameter `fields`. The value of the parameter is a comma separated list of filter IDs to be called.

For example:

```
...
<services>
  <service type="webfeed" id="feed">
    ...
    <reference from="fieldsparam" to="fields"/>
    ...
  </service>
...
</services>
...
<parameters>
```

The service

Webfeed_service

```
<parameter type="string"
  name="fieldsparam"
  editable="false"
  visible="false"
  description="Fields to extract and show">author,created,title,content</parameter>
...
</parameters>
...
```

The filters are called in the order they appear in the list. The order specifies also the order of fields in the data structure returned to the client. It is a good practice to keep the filter IDs the same as the resulting field names.

Note that the webfeed service adds the `link` field automatically to the list of fields even if `link` is not listed in the `fields` parameter.

The filters referred to in the `fields` parameter are either custom filters defined in the `<filters>` section of *widget.xml* or default filters. If a custom filter definition is not found, then a default filter is used if one exists. There are default filters defined for the following fields:

- author
- content
- created
- link
- title

Main filter

The default main filter is capable of recognizing three different feed types:

- RSS (RSS 2.0)
- RDF (RSS 0.9, RSS 1.0)
- ATOM

Below is the definition of the default main filter:

```
<filter id="main">
  <choice>
    <list>
      <foreach>
        <xpath>/rss/channel/item</xpath>
        <list>
          <name>
            <hook id="rssitemid"/>
          </name>
          <value>
            <hook id="rssitemfields"/>
          </value>
        </list>
      </foreach>
    </list>
    <list>
      <foreach>
        <xpath>/rdf:RDF/channel/items/rdf:Seq/rdf:li</xpath>
        <xpath>
          <strcat>
```

Webfeed_service

```
<str>/rdf:RDF/item[@rdf:about='</str>
<xpath>./@rdf:resource</xpath>
<str>'] [1]</str>
</strcat>
</xpath>
<list>
  <name>
    <hook id="rdfitemid"/>
  </name>
  <value>
    <hook id="rdfitemfields"/>
  </value>
</list>
</foreach>
</list>
<list>
  <foreach>
    <xpath>/feed/entry</xpath>
    <list>
      <name>
        <hook id="atomitemid"/>
      </name>
      <value>
        <hook id="atomitemfields"/>
      </value>
    </list>
  </foreach>
</list>
</choice>
</filter>
```

The filter consists of a `choice` expression with three optional branches. In practice, the `xpath` expressions within the `foreach` expressions' values-parts handle recognizing the different feed types.

The structure of each branch is basically the same (the RDF branch being more complex than the rest). Each branch produces a nameless list into which the `foreach` expression generates the elements. Each `foreach` expression uses as a values-part an `xpath` expression to extract the item constructs from the input feed xml-file, and loops through the item constructs to produce the feed item elements. The do-part of each `foreach` expression generates an item, that is., a named list of item field elements from each item construct. The name of the list is the ID of the item.

The `hook` expressions within the do-parts of the `foreach` expressions are links to the filters that are called from the main filter. The `hook` expression named as "`...itemid`" calls the item ID filter that produces the ID of each item. The `hook` expression named as "`...itemfields`" calls the field generating filters, that is, the filters listed in the `fields` service parameters plus the `link` field filter.

Currently the webfeed service relies on naming conventions when linking field filters to the main filter. The IDs of the `hook` expressions are fixed.

The item ID filter hooks are linked as follows:

Hook ID...	... hooks to Filter ID
rssitemid	rssitems.itemid or, if that is not defined, itemid
rdfitemid	rdffitems.itemid or, if that is not defined, itemid

Webfeed_service

atomitemid	atomitems.itemid or, if that is not defined, itemid
------------	---

The fields filter hooks are linked as follows: below `field` is a field name listed in the `fields` service parameter:

Hook ID...	... hooks to Filter IDs
rssitemfields	rssitems. <i>field</i> or, if that is not defined, <i>field</i>
rdfitemfields	rdfitems. <i>field</i> or, if that is not defined, <i>field</i>
atomitemfields	atomitems. <i>field</i> or, if that is not defined, <i>field</i>

Overriding default field filters

Field filters generate the field values of feed items. A field value is a named list or item element created by `list` or `item` filter expression, respectively. A field filter gets as input an item XML construct extracted by the main filter. The common fields `author`, `created`, `content`, `title`, and `link` have predefined default filters for each feed type (RSS, RDF, and ATOM).

For example, the default `title` field filter for RSS feeds look like the following:

```
<filter id="rssitems.title">
  <item name="title">
    <choice>
      <xpath>title/text()</xpath>
      <xpath>dc:title/text()</xpath>
    </choice>
    <untag/>
  </item>
</filter>
```

The filter gets as an input an item XML construct from the main filter. The filter generates a named item element, named as "title". The value of the named item element is the value of the 'title' XML element or the 'dc:title' XML element of the input item construct. The value is also "untagged" from any html tags with the `untag` filter expression.

Each of the default filters can be overridden. A default field filter is overridden by defining a filter with the same filter ID in the `filters` section of the `widget.xml` file. Because the main filter has different branches for RSS, RDF, and ATOM feeds, there are RSS, RDF, and ATOM specific filters for each field. An overriding field filter can override the field filter of a specific feed type or the field filters of all feed types. If the ID of the overriding filter is the plain field name like `title`, then the filter overrides the field filters of all the feed types. To override the field filter of only one of the feed types, the filter ID must contain a prefix that specifies the feed type.

The available prefixes are:

Feed Type	Filter Prefix
RSS	rssitems.

Webfeed_service

RDF	<code>rdfitems.</code>
ATOM	<code>atomitems.</code>

To override the default filter for the title field of ATOM feeds, define a filter with the ID `atomitems.title`. To override the default filter for the title field of all the feed types, define a filter with the ID `title`.

Note that this RSS-RDF-ATOM separation is necessary only if the custom widget needs to support multiple feed types. If the widget is targeted for a specific feed or feed type, there is no reason to use the filter ID prefixes or worry about webfeed service internals: it is enough to use plain field names as the filter IDs.

Note also that only the strings `rssitems.`, `rdfitems.`, and `atomitems.` are handled as special prefixes. Other dot-separated strings are treated as part of the filter IDs. As a result, a filter ID like `digg.username` behaves exactly the same as `digg_username`.

Overriding item ID filter

Item ID filter creates the ID of a feed item. Item ID filter differs from field filters so that it is always called for each item and that it should return a plain (string) value instead of list or item element. As with the field filters, there are three separate item ID filters, one for each feed type RSS, RDF, and ATOM. All tree default item ID filters are identical, though. The default item ID filter is the following:

```
<filter id="itemid">
  <choice>
    <xpath>guid/text()</xpath>
    <xpath>id/text()</xpath>
    <xpath>link/text()</xpath>
  </choice>
  <to_hash/>
</filter>
```

The default item ID filter gets as input the item construct the main filter has extracted and tries to extract the value of the `guid`, `id`, or `link` XML element. It then uses the `to_hash` filter expression to compute an MD5 hash value of the element value as a hex value string. This hash value string is the result of the filter and will be the ID of the item.

The default item ID filters of the different feed types can be overridden like the field filters by defining a filter with the ID `itemid` and optionally using prefixes `rssitems.`, `rdfitems.`, and `atomitems.`

Paged fields

The webfeed service supports field value *paging*. The client does not have to retrieve the full content of a potentially long item field, but can get the content page by page. The `getItems` service action returns the first page of all paged fields (and the full content of all non-paged fields). The `getPage` service action can be used to retrieve the additional pages of a paged item field.

By default, only the `content` field is paged, but any field can be marked as paged by following a special convention in the field generating filter. A paged field is generated with a filter that has the following structure:

Webfeed_service

```
<filter>
  <list name=field-name>
    <item name="type" value="paged"/>
    <list name="val">
      expressions-generating-the-full-field-value
    </list>
  </list>
</filter>
```

A paged field is a named list that contains two special elements: the *type element* and the *value element*. The name of the list is the name of the field.

The type element specifies that the field is paged. The type element is an item element with the name "type" and the value "paged":

```
<item name="type" value="paged"/>
```

The value element contains the filter expressions that generate the actual (full) value of the field. The value element is a list element with the name "val":

```
<list name="val">
  expressions-generating-the-full-field-value
</list>
```

The field value is generated by the filter expressions in the value part of the list element. The expressions should generate zero or more item or list elements. Item elements are used to represent text elements of the content. List elements are used to represent image url elements.

Page and content lengths are counted as number of characters. An image url element is counted as one character.

When a paged field is sent to the client, the filter generated structure is not sent as is, instead the webfeed service extracts the requested portion of the value (the page) and forms the following output structure:

```
field-name = (type = page
              pg  = page-number
              left = number-of-pages-left
              val  = (actual-page-value))
```

Note that the `type` attribute in the output is different from the `type` attribute in the original filter. Even the values are different: `page` vs. `paged`. The attribute in the filter tells the webfeed service to do paging on the field value. The attribute in the output tells the client the the field value is of type `page`.

Example

Below is the default filter for the `content` field of RSS feed items. The filter takes, for example, the string value of the `description` RSS feed element and splits the value to a list of string and image url elements:

```
<filter id="rssitems.content">
  <choice>
    <xpath>content:encoded/text ()</xpath>
    <xpath>description/text ()</xpath>
    <xpath>div/text ()</xpath>
    <xpath>body/text ()</xpath>
```

Webfeed_service

```
<xpath>dcterms:abstract/text ()</xpath>
<xpath>dc:description/text ()</xpath>
<xpath>source/text ()</xpath>
</choice>

<var id="regex_input">
  <get/>
</var>

<list name="content">
  <item name="type" value="paged"/>
  <list name="val">
    <choice>
      <regex>
        <![CDATA[
          (?s)<img(?:[ ]+|(?:[ ]+[>]*[ ]+))src=[""]?([" '"]*)[" "].*?>
        ]]>
      </regex>
      <regex>
        <![CDATA[
          (?s)$
        ]]>
      </regex>
    </choice>

    <var id="previous" value="0"/>

    <iterator>
      <next>
        <item>
          <substr>
            <source>
              <get id="regex_input"/>
            </source>
            <start value="{previous}"/>
            <end>
              <regex_match_start/>
            </end>
          </substr>
          <entity_decode/>
          <untag/>
          <str_trim min_length="1"/>
        </item>

        <list>
          <choice>
            <regex_match group="1"/>
          </choice>
          <item name="type" value="image"/>
          <item name="url">
            <get/>
          </item>
        </list>

        <regex_match_end to="previous"/>
      </next>

      <after_last>
        <item>
          <substr>
            <source>
              <get id="regex_input"/>
            </source>
          </substr>
        </item>
      </after_last>
    </iterator>
  </list>
</list>
```

Webfeed_service

```
        </source>
        <start value="{previous}"/>
    </substr>
    <entity_decode/>
    <untag/>
    <str_trim min_length="1"/>
</item>
</after_last>
</iterator>
</list>
</list>
</filter>
```

Specifying custom field filters

Custom field filters are used to generate custom field values of feed items. Like the predefined field values, a custom field value is a named list or item element created by `list` or `item` filter expression. Like the default field filters, a custom field filter also gets as input an item XML construct extracted by the main filter.

Specifying custom field filters is easy: just define the filter in `widget.xml` file and add the filter ID to the value of the `fields` service parameter. The filter then gets linked to and called by the main filter.

For example, to extract the Digg username from a digg feed, define the following filter:

```
<filter id="digg.username">
  <item name="digg.username">
    <xpath>digg:submitter/digg:username/text ()</xpath>
  </item>
</filter>
```

The ID of the filter `digg.username` should then be added to the value of `fields` service parameter to get it linked to the main filter.

The ID of a custom field filter can be freely chosen - only the predefined field names `author`, `created`, `content`, `title`, and `link` cannot be used, unless you want to override a default filter.

Advanced: overriding main filter

To use webfeed service to parse and handle something else than RSS, RDF, or ATOM feed, you must override the main filter. The main filter can be overridden simply by defining a filter with the ID `main`.

The current webfeed service supports only a fixed set of hook IDs to link field filters to the main filter ([see: Main filter](#)). This applies also to a custom main filter. Therefore you must pick one or several of the available hook IDs to be used in your custom main filter, event if the hook ID names do not make any sense. For example, you can use the hook ID `rssitemid` to link to your item ID filter, and the hook ID `rssitemfields` to link your field filters. Then you can name your item id filter either as `rssitems.itemid` or simply as `itemid` and your field filters either as `rssitems.field` or simply as `field`.

Note. There are plans to enhance the webfeed service so that one could use custom hook IDs.

Webfeed_service

For example, here is a main filter that can parse a [NewsML](#) file:

```
<filter id="main">
  <list>
    <foreach>
      <xpath>/NewsML/NewsItem/NewsComponent/NewsItem</xpath>
      <list>
        <name>
          <hook id="rssitemid"/>
        </name>
        <value>
          <hook id="rssitemfields"/>
        </value>
      </list>
    </foreach>
  </list>
</filter>
```

The following are custom filters to generate item ID and the field headline:

```
<filter id="itemid">
  <xpath>Identification/NewsIdentifier/NewsItemId/text () </xpath>
  <to_hash/>
</filter>

<filter id="headline">
  <item name="headline">
    <xpath>NewsComponent/NewsLines/HeadLine/text () </xpath>
  </item>
</filter>
```

The headline filter ID must then be listed in the `fields` service parameter value in order to be called from the main filter.

Service Parameters

The XML configuration file of a webfeed service can define several service parameters that define and control the service.

Service parameter	Type	Meaning
feedurl	mandatory	The URL to feed to be fetched.
fields	mandatory	The fields to include to the output. More specifically: the field filters to apply when parsing the feed.
polltype	optional	Marks the service as pollable. The value of this parameter defines how the server will poll the feed defined by the <code>feedurl</code> parameter. Currently the only valid value for this parameter is <code>auto</code> .
<code>minpollperiod</code>	optional	Defines the lower limit for the poll period value for a pollable service. Default value is "2m".
<code>maxpollperiod</code>	optional	

Webfeed_service

		Defines the upper limit for the poll period value for a pollable service. Default value is "3h".
authtoken	optional	TOKEN to be used in signing the URL when Token authentication is used. (See Token authentication .)
userid	optional	UserID to be used when performing Token authentication. (See Token authentication .)
partnerid	optional	Id of the partner in question when using Token authentication.
useapiurl	optional	If the value is true the URL specified in the partner's information will be used. Otherwise the URL specified by feedurl service parameter will be used.

Service actions

The service has four actions the client script can call.

Action	Generic	Function
<u>getItems</u>	No	Retrive feed items list.
<u>getPage</u>	No	Retrieve the value of a field page by page. The default page length is 500 characters.
<u>bookmark</u>	No	Save the link associated with a feed item into the bookmarks.
<u>senditem</u>	No	Send the link associated with a feed item to a friend by email.

Below are the specifications of the service actions.

Definitions

```
FieldVal      = (choice (null noData)
                   (int   intVal)
                   (string stringVal)
                   (use   ImageUrlVal)
                   (use   ElemList))

PagedFieldVal = (list (bag (bind (const type) (const page))
                           (bind (const pg)  (int   pageNum))
                           (bind (const left) (int   pagesLeft))
                           (bind (const val)  (use   PageVal))))

PageVal       = (choice (null noData)
                   (use   ElemList))

ElemList      = (list (repeat (use Elem)))

Elem          = (choice (string stringVal)
                   (use   ImageUrlVal))
```

Webfeed_service

```
ImageUrlVal = (list (bag (required (bind (const type) (const image)))
                        (required (bind (const url) (string imageUrl)))
                        (optional (bind (const width) (int imageWidth)))
                        (optional (bind (const height) (int imageHeight)))))
```

Action getItems

Retrieve feed items list.

```
input = (list (bag (optional (bind (const ts) (int timestamp)))
                (optional (bind (const refresh) (boolean refreshVal)))
                (optional (bind (const max) (int maxCount)))
                (optional (bind (const len) (choice (int pageLength)
                                                    (const all))))))

output = (choice (null noData)
              (list (repeat (use Item))))

Item = (list (bag (bind (const iid) (string itemId))
               (repeat (use Field))))

Field = (bind (string fld) (choice (use FieldVal)
                                   (use PagedFieldVal)))
```

Arguments

Name	Type	Description
ts	int	Return only items that are newer than the value of this argument. The timestamp value is in milliseconds since the 1st of January, 1970, and interpreted as UTF/GMT timestamp.
refresh	boolean	If true, return fresh content fetched from the feedurl. Otherwise, return content from server side cache if available.
max	int	The maximum number of items to return. Default is 20.
len	int all	The [first] page length (in characters) for the fields marked as paged. The content field is paged by default. Default page length is 500. The special constant value all returns all content.

Action getPage

Retrieve the value of the content field page by page.

```
input = (list (bag (required (bind (const iid) (string itemId)))
                 (optional (bind (const fld) (string fieldName)))
                 (required (bind (const pg) (int pageNum)))
                 (optional (bind (const len) (choice (int pageLength)
                                                    (const all))))))
```

Webfeed_service

```
output = (list (bag (bind (const iid) (string itemId))
                    (use PagedField)))

PagedField = (bind (string fld) (choice (null noData)
                                       (use PagedFieldVal)))
```

Arguments

Name	Type	Description
iid	string	The ID of the item.
pg	int	The number of the page to return. Page numbers start from 1.
fld	string	The name of the field. <code>content</code> if not specified.
len	int all	The page length (in characters). Default is 500. The special constant value <code>all</code> returns all content. (If <code>len</code> is <code>all</code> then <code>pg</code> must be 1.)

Action bookmark

Save the link associated with a feed item into the bookmarks.

```
input = (list (bag (required (bind (const type) (choice (const default)
                                                         (const custom))))
                (optional (bind (const iid) (string itemId)))))

output = (const true)
```

Arguments

Name	Type	Description
type	default custom	Type of the bookmark, normally default.
iid	string	The ID of the item to bookmark.

Action senditem

Send the link associated with a feed item to a friend by email.

```
input = (list (bag (required (bind (const iid) (string itemId))
                                (required (bind (const to) (string to)))))

output = (const true)
```

Arguments

Name	Type	Description
iid	string	The ID of the item to send.
to	string	The recipient's email address.

Example

Below is an example of a webfeed widget definition. Note that only parts of the widget.xml are shown.

The widget parses RSS feed from [Digg](#). The widget introduces three custom fields: `diggCount`, `category`, and `avatar`. The widget overrides the default filter for the `author` field. For the rest of the fields -- `title`, `created`, `content`, and `link` -- the widget uses the predefined default filters.

Note. The widget defines the `image` service so that the client can retrieve and show the avatar images. The widget defines the `polltype` parameter so that the server will poll the feed automatically.

```
...
<services>
  <service type="webfeed" id="feed1">
    <reference from="feed1" to="feedurl"/>
    <reference from="fieldsparam" to="fields"/>
    <reference from="polltypeparam" to="polltype"/>
  </service>
  <service type="image" id="img"/>
</services>
...
<parameters>
  <parameter name="feed1"
    type="string"
    description="Feed url"
    editable="true"
    protected="false"
    visible="true"
    sendmobile="true">
    <value>http://www.digg.com/rss/index.xml</value>
  </parameter>

  <parameter type="string"
    name="fieldsparam"
    editable="false"
    visible="false"
    description="Fields to extract and show">
    <value>category,diggCount,title,avatar,created,author,content</value>
  </parameter>

  <parameter name="polltypeparam"
    type="string"
    description=""
    editable="true"
    protected="false"
    visible="false"
    sendmobile="false">
```

Webfeed_service

```
    <value>auto</value>
  </parameter>
  ...
</parameters>
...
<filters>
  <filter id="diggCount">
    <item name="diggCount">
      <strcat>
        <xpath>digg:diggCount/text ()</xpath>
      </strcat>
    </item>
  </filter>

  <filter id="author">
    <item name="author">
      <xpath>digg:submitter/digg:username/text ()</xpath>
    </item>
  </filter>

  <filter id="avatar">
    <item name="avatar">
      <xpath>digg:submitter/digg:userimage/text ()</xpath>
    </item>
  </filter>

  <filter id="category">
    <item name="category">
      <xpath>digg:category/text ()</xpath>
    </item>
  </filter>
</filters>
...

```

The following is sample output from the `getItems` action of the above webfeed service:

```
(
(iiid = fe38d315641555e2026a2d47fe1de9a1
 category = Space
 diggCount = "87"
 title = "Rosetta: Earth\u2019s true colours [pics]"
 avatar = "http://digg.com/users/cosmikdebris/m.png"
 created = 1195731100000
 author = cosmikdebris
 content = (type = page
            pg = 1
            left = 0
            val = ("True colour images of Earth as seen by Rosetta\u2019s OSIRIS camera are now
                    available. The pictures were taken on 13 November during the swing-by, and on
                    15 November, as Rosetta left on its way to the outer Solar System, after the
                    swing-by."))
 link = "http://digg.com/space/Rosetta_Earth_s_true_colours_pics")

(iiid = "3d6d28d29f42552cafe5f6d7ff02486a"
 category = "Tech Industry News"
 diggCount = "233"
 title = "Interesting Web Browsers You Have Never Heard Of"
 avatar = "http://digg.com/img/udm.png"
 created = 1195725603000
 author = scbalazs
 content = (type = page
            pg = 1

```

Webfeed_service

```
        left = 0
        val = ("A quick review of some new (and old) browsers as alternatives.")
link = "http://digg.com/tech_news/Interesting_Web_Browsers_You_Have_Never_Heard_Of_2")

(iiid = fa11003fb42564b8689ac4d16abb4e9a
category = Linux/Unix
diggCount = "98"
title = "Embedded Linux Goes Open Source"
avatar = "http://digg.com/users/Bindibadgi/m.png"
created = 1195725602000
author = Bindibadgi
content = (type = page
           pg = 1
           left = 0
           val = ("The embedded Linux on the Asus P5E3 Deluxe WiFi-AP @n motherboard
                  has just had its source code released and bit-tech has talked to Asus
                  and the guys that make it to find out what they expect this will mean
                  for future products. Will the dev community jump on board? What about
                  boards that play music and video and boot in 2 seconds?"))
link = "http://digg.com/linux_unix/Embedded_Linux_Goes_Open_Source")

...
)
```

See also

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