Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# EiffelRSS

*SYNDICATION Developer Guide*

Michael Käser <kaeserm@student.ethz.ch>
Martin Luder <luderm@student.ethz.ch>
Thomas Weibel <weibelt@student.ethz.ch>

Informatik
Computer Science

**Abstract**

SYNDICATION is the main cluster of EiffelRSS with a feed object model and classes to load / write feeds. It is divided into three subclusters.

# Contents

# List of Figures

# Part I

# INTERFACE

# Chapter 1

# Overview

INTERFACE is the sub-cluster of syndication with all the classes a developer needs to use the library. There are classes to read into and write from a FEED, a FEED_MANAGER to administrate a list of FEEDs, and a factory class which makes it easy to create all necessary objects.

See figure 1.1 for an overview of the cluster.

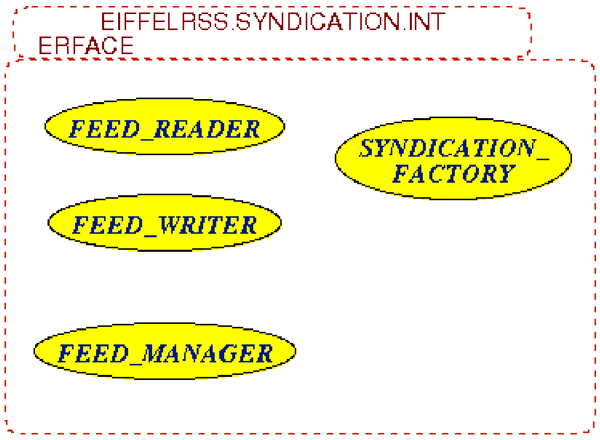

**Figure 1.1:** BON diagram of cluster INTERFACE

# Chapter 2

# Class SYNDICATION_FACTORY

## 2.1 Overview

`SYNDICATION_FACTORY` provides an easy way to create objects of classes from the cluster `SYNDICATION`.

## 2.2 Usage

```
class USAGE_EXAMPLE

create
  make

feature -- Initialization

  make is
      -- Creation procedure.
    do
      create syndication

      feed := syndication.new_feed ("EiffelRSS", create ↘
      →{HTTP_URL}.make ("http://eiffelrss.berlios.de/"),↘
      → "EiffelRSS news")

      item := syndication.new_item (channel, "Version 23↘
      → released!", create {HTTP_URL}.make ("http://↘
      →eiffelrss.berlios.de/Main/News"), "Version 23 of ↘
      →EiffelRSS got release today. Happy syndicating!")
      feed.add_item (item)
    end

feature -- Arguments

  syndication: SYNDICATION_FACTORY
      -- Syndication factory object

  feed: FEED
      -- Feed object

  item: ITEM
      -- Item object

end -- class USAGE_EXAMPLE
```

## 2.3 Features

### 2.3.1 READER factory

**new_reader_from_url**

```
new_reader_from_url (a_url: STRING): FEED_READER
  -- Create with 'a_url' as source of feed
```

### 2.3.2 WRITER factory

**new_writer_from_feed**

```
new_writer_from_feed (a_feed: FEED): FEED_WRITER
  -- Create a writer object for the feed 'a_feed'
```

### 2.3.3 FEED_MANAGER factory

**new_feed_manager**

```
new_feed_manager: FEED_MANAGER
  -- Create a new feed manager with default refresh ↘
  →period '30'
```

**new_feed_manager_custom**

```
new_feed_manager_custom (a_refresh_period: INTEGER): ↘
→FEED_MANAGER
  -- Create a new feed manager with default refresh ↘
  →period 'a_refresh_period'
```

### 2.3.4 FEED factory

**new_feed**

```
new_feed (a_title: STRING; a_link: URL; a_description: ↘
→STRING): FEED
  --  Create a feed with title, link and description
```

**new_feed_from_channel**

```
new_feed_from_channel (a_channel: CHANNEL): FEED
  -- Create a new feed from an existing channel
```

### 2.3.5 CHANNEL factory

**new_channel**

```
new_channel (a_title: STRING; a_link: URL; a_description↘
→: STRING): CHANNEL
  --  Create a channel with title, link and description
```

**new_channel_cloud**

```
new_channel_cloud (a_domain: STRING; a_port: INTEGER; ↘
→a_path: STRING; a_register_procedure: STRING; ↘
→a_protocol: STRING): CHANNEL_CLOUD
  -- Create a channel cloud with domain, port, path, ↘
  →register procedure and protocol
```

#### new_channel_image

```
new_channel_image (a_url: URL; a_title: STRING; a_link: ↘
→URL): CHANNEL_IMAGE
  -- Create a channel image with URL, title, and link
```

#### new_channel_text_input

```
new_channel_text_input (a_title: STRING; a_description: ↘
→STRING; a_name: STRING; a_link: URL): ↘
→CHANNEL_TEXT_INPUT
  -- Create a channel text input with title, description↘
  →, name and link
```

### 2.3.6 ITEM factory

#### new_item

```
new_item (a_channel: CHANNEL; a_title: STRING; a_link: ↘
→URL; a_description: STRING): ITEM
  -- Create an item with title, link and description
```

#### new_item_with_title

```
new_item_with_title (a_channel: CHANNEL; a_title: STRING↘
→): ITEM
  -- Create an item with title
```

#### new_item_with_description

```
new_item_with_description (a_channel: CHANNEL; ↘
→a_description: STRING): ITEM
  -- Create an item with description
```

#### new_item_enclosure

```
new_item_enclosure (a_url: URL; a_length: INTEGER; ↘
→a_type: STRING): ITEM_ENCLOSURE
  -- Create an item enclosure
```

**new_item_guid**

```
new_item_guid (a_guid: STRING): ITEM_GUID
  -- Create an item guid with 'is_perma_link' set to ↘
  →False
```

**new_item_guid_perma_link**

```
new_item_guid_perma_link (a_guid: STRING): ITEM_GUID
  -- Create an item guid with 'is_perma_link' set to ↘
  →True
```

**new_item_source**

```
new_item_source (a_name: STRING; a_url: URL): ↘
→ITEM_SOURCE
  -- Create an item source
```

### 2.3.7 CATEGORY factory

**new_category**

```
new_category: CATEGORY
  -- Create a category with title '[unnamed category]')
```

**new_category_with_title**

```
new_category_with_title (a_title: STRING): CATEGORY
  -- Create a category with title 'a_title'
```

**new_category_with_title_domain**

```
new_category_with_title_domain (a_title: STRING; ↘
→a_domain: URL): CATEGORY
  -- Create a category with title 'a_title' and domain '↘
  →a_domain'
```

# Chapter 3

# Class FEED_MANAGER

## 3.1 Overview

FEED_MANAGER is a class to manage feeds. It provides features to add, remove and refresh feeds.

See figure 3.1 for an overview of the class.

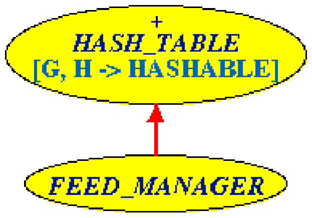

**Figure 3.1:** BON diagram of class FEED_MANAGER

## 3.2 Usage

```
class
  FEED_MANAGER_EXAMPLE

create
  make

feature -- Initialization

  make is
      -- Creation procedure.
    do
      -- Create a simple feed
```

```
            create feed.make ("EiffelRSS", create {HTTP_URL}.↘
            →make ("http://eiffelrss.berlios.de"), "EiffelRSS ↘
            →news")
            feed.set_refresh_period (15)
            feed.set_last_updated (create {DATE_TIME}.make_now↘
            →)

            -- Add some simple items, use 'feed.↘
            →last_added_item' or directly create an item for ↘
            →finer control
            feed.new_item ("Version 23 released!", create {↘
            →HTTP_URL}.make ("http://eiffelrss.berlios.de/Main↘
            →/News"), "Version 23 of EiffelRSS got release ↘
            →today. Happy syndicating!")

            feed.new_item ("EiffelRSS wins award", create {↘
            →HTTP_URL}.make ("http://eiffelrss.berlios.de/Main↘
            →/Awards"), "EiffelRSS has been awarded by ISE as ↘
            →best syndication software written in Eiffel. For ↘
            →more info see award-winning pages: http://↘
            →eiffelrss.berlios.de")

            -- Create feed manager
            create feed_manager.make
            feed_manager.add (feed, "http://eiffelrss.berlios.↘
            →de/Main/AllRecentChanges?action=rss")
            feed_manager.refresh_all
        end

feature -- Arguments

    feed: FEED
            -- Example feed

    feed_manager: FEED_MANAGER
            -- Feed manager

end -- class FEED_MANAGER_EXAMPLE
```

## 3.3 Features

### 3.3.1 Initialization

#### make

```
make
    -- Create a new feed manager with default refresh ↘
    →period '30'
```

#### make_custom

```
make_custom (a_refresh_period: INTEGER)
    -- Create a new feed manager with default refresh ↘
    →period 'a_refresh_period'
```

### 3.3.2 Access

#### default_refresh_period

```
default_refresh_period: INTEGER
    -- Default refresh period in minutes
```

#### last_added_feed

```
last_added_feed: FEED
    -- feed that was last added
```

#### feed_addresses

```
feed_addresses: LINKED_LIST [STRING]
    -- Returns a sortable list representation of the ↘
    →feeds saved in FEED_MANAGER
```

#### feed_links

```
feed_links: LINKED_LIST [STRING]
    -- Returns a sortable list representation of the ↘
    →feeds saved in FEED_MANAGER
```

### 3.3.3 Setter

**set_default_refresh_period**

```
set_default_refresh_period (a_refresh_period: INTEGER)
    -- Set refresh periode in minutes
```

### 3.3.4 Element change

**add**

```
add (feed: FEED; url: STRING)
    -- Add `feed'
```

**add_from_url**

```
add_from_url (url: STRING)
    -- Add feed with URL `url'
```

### 3.3.5 Refresh

**refresh**

```
refresh (url: STRING)
    -- Refresh feed with URL `url', if the feed is ↘
    →outdated
```

**refresh_force**

```
refresh_force (url: STRING)
    -- Refresh feed with URL `url', even if the feed is ↘
    →not outdated
```

**refresh_all**

```
refresh_all
    -- Refresh all feeds, if they are outdated
```

**refresh_all_force**

```
refresh_all_force
    -- Refresh all feeds, even if they are not outdated
```

### 3.3.6 Conversion

**list_representation**

```
list_representation: SORTABLE_TWO_WAY_LIST[FEED]
    -- Returns a sortable list representation of the ↘
    →feeds saved in FEED_MANAGER
```

### 3.3.7 Conversion (sort)

**sorted_by_last_updated**

```
sorted_by_last_updated: SORTABLE_TWO_WAY_LIST[FEED]
    -- Returns a sorted list representation of the feeds↘
    →, sorted by 'last_updated'
```

**sorted_by_title**

```
sorted_by_title: SORTABLE_TWO_WAY_LIST[FEED]
    -- Returns a sorted list representation of the feeds↘
    →, sorted by 'title'
```

**sorted_by_link**

```
sorted_by_link: SORTABLE_TWO_WAY_LIST[FEED]
    -- Returns a sorted list representation of the feeds↘
    →, sorted by 'link'
```

**sorted_by_description**

```
sorted_by_description: SORTABLE_TWO_WAY_LIST[FEED]
    -- Returns a sorted list representation of the feeds↘
    →, sorted by 'description'
```

**reverse_sorted_by_last_updated**

```
reverse_sorted_by_last_updated: SORTABLE_TWO_WAY_LIST[↘
→FEED]
    -- Returns a sorted list representation of the feeds↘
    →, reverse sorted by 'last_updated'
```

**reverse_sorted_by_title**

```
reverse_sorted_by_title: SORTABLE_TWO_WAY_LIST[FEED]
    -- Returns a sorted list representation of the feeds↘
    →, reverse sorted by 'title'
```

**reverse_sorted_by_link**

```
reverse_sorted_by_link: SORTABLE_TWO_WAY_LIST[FEED]
    -- Returns a sorted list representation of the feeds↘
    →, reverse sorted by 'link'
```

**reverse_sorted_by_description**

```
reverse_sorted_by_description: SORTABLE_TWO_WAY_LIST[↘
→FEED]
    -- Returns a sorted list representation of the feeds↘
    →, reverse sorted by 'description'
```

# Chapter 4

# Class FEED_READER

## 4.1 Overview

FEED_READER is a helper class which manages everything to load a feed. It converts the data to an XML document object, detects the format of the feed and uses the according reader object to convert the XML document into a FEED object.

See figure 4.1 for an overview of the class.

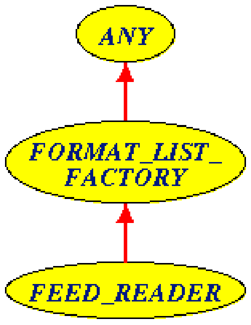

**Figure 4.1:** BON diagram of class FEED_READER

## 4.2 Usage

```
class
   READER_EXAMPLE

create
   make

feature -- Initialization
```

```
  make is
      -- Creation procedure.
    local
      location: STRING
      reader: FEED_READER
      feed: FEED
    do
      -- Get a feed location from the user
      io.put_string ("Enter an URL: ")
      io.read_line

      location := io.last_string.twin

      -- Create the reader
      create reader.make_url (location)

      -- Get the feed
      feed := reader.read

      -- Print feed
      io.put_string ("%NReceived feed:%N")
      io.put_string ("=============%N%N%N")
      io.put_string (feed.to_string)
    end

end -- class READER_EXAMPLE
```

## 4.3 Features

### 4.3.1 Initialization

**make_url**

```
make_url (a_url: STRING)
  -- Create with 'a_url' as source of feed
```

### 4.3.2 Basic operations

**read**

```
read: FEED
  -- Load the data from the given url into a FEED
```

# Chapter 5

# Class FEED_WRITER

## 5.1 Overview

`FEED_WRITER` is a helper class which manages everything to write a feed. It converts the data from an existing `FEED` object into an XML document object and saves it into a local file.

## 5.2 Usage

```
class
  WRITER_EXAMPLE

create
  make

feature -- Initialization

  make is
      -- Creation procedure.
    local
      feed: FEED
      writer: FEED_WRITER
    do
      -- Create a simple feed
      create feed.make ("EiffelRSS", create {HTTP_URL}.↘
      →make ("http://eiffelrss.berlios.de/Main/↘
      →AllRecentChanges?action=rss"), "EiffelRSS news")

      -- Add some simple items
      feed.new_item ("Version 23 released!", create {↘
      →HTTP_URL}.make ("http://eiffelrss.berlios.de/Main↘
            /News"), "Version 23 of EiffelRSS got release
            today. Happy syndicating!")
            feed.new_item ("Microsoft uses EiffelRSS", create
            {HTTP_URL}.make ("http://eiffelrss.berlios.de/
            Main/WhoUsesEiffelRSS"), "Microsoft announced in
            a press release today that they will use
            EiffelRSS to syndicate news on their website.")
            feed.new_item ("EiffelRSS wins award", create {
            HTTP_URL}.make ("http://eiffelrss.berlios.de/Main
            /Awards"), "EiffelRSS has been awarded by ISE as
            best syndication software written in Eiffel. For
            more info see award-winning pages: http://
            eiffelrss.berlios.de")

            -- Write feed to file
            create writer.make_feed (feed)
            writer.write ("example.xml", "RSS 2.0")
    end

end -- class WRITER_EXAMPLE
```

## 5.3 Features

### 5.3.1 Initialization

**make_feed**

```
make_feed (a_feed: FEED) is
  -- Create a writer object for the feed 'a_feed'
```

### 5.3.2 Basic operations

**write**

```
write (a_filename, a_format: STRING) is
  -- Write the feed to a local file with 'a_filename' in
  the format 'a_format'
  -- You can enumerate all available formats with
  FORMAT_LIST (see FORMATS)
```

# Part II

# FEED

# Chapter 6

# Overview

FEED is the central datastructure of EiffelRSS. It defines an abstract syndication feed.

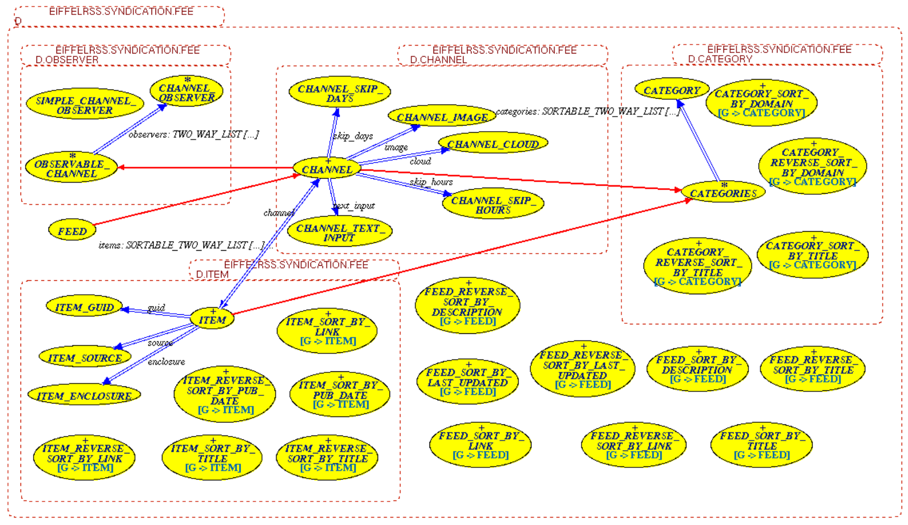
EIFFELRSS.SYNDICATION.FEED
EIFFELRSS.SYNDICATION.FEED.OBSERVER
SIMPLE_CHANNEL_OBSERVER
CHANNEL_OBSERVER
OBSERVABLE_CHANNEL
observers: TWO_WAY_LIST [...]
EIFFELRSS.SYNDICATION.FEED.CHANNEL
CHANNEL_SKIP_DAYS
CHANNEL_IMAGE
CHANNEL_CLOUD
CHANNEL
CHANNEL_SKIP_HOURS
CHANNEL_TEXT_INPUT
skip_days
image
cloud
skip_hours
text_input
categories: SORTABLE_TWO_WAY_LIST [...]
EIFFELRSS.SYNDICATION.FEED.CATEGORY
CATEGORY
CATEGORY_SORT_BY_DOMAIN [G -> CATEGORY]
CATEGORY_REVERSE_SORT_BY_DOMAIN [G -> CATEGORY]
CATEGORIES
CATEGORY_REVERSE_SORT_BY_TITLE [G -> CATEGORY]
CATEGORY_SORT_BY_TITLE [G -> CATEGORY]
FEED
channel
items: SORTABLE_TWO_WAY_LIST [...]
EIFFELRSS.SYNDICATION.FEED.ITEM
ITEM_GUID
guid
ITEM
ITEM_SOURCE
source
enclosure
ITEM_ENCLOSURE
ITEM_SORT_BY_LINK [G -> ITEM]
ITEM_REVERSE_SORT_BY_PUB_DATE [G -> ITEM]
ITEM_SORT_BY_PUB_DATE [G -> ITEM]
ITEM_REVERSE_SORT_BY_LINK [G -> ITEM]
ITEM_SORT_BY_TITLE [G -> ITEM]
ITEM_REVERSE_SORT_BY_TITLE [G -> ITEM]
FEED_REVERSE_SORT_BY_DESCRIPTION [G -> FEED]
FEED_SORT_BY_LAST_UPDATED [G -> FEED]
FEED_REVERSE_SORT_BY_LAST_UPDATED [G -> FEED]
FEED_SORT_BY_DESCRIPTION [G -> FEED]
FEED_REVERSE_SORT_BY_TITLE [G -> FEED]
FEED_SORT_BY_LINK [G -> FEED]
FEED_REVERSE_SORT_BY_LINK [G -> FEED]
FEED_SORT_BY_TITLE [G -> FEED]

**Figure 6.1:** BON diagram of cluster FEED

# Chapter 7

# Usage

```
class
  FEED_EXAMPLE

create
  make

feature -- Initialization

  make is
      -- Creation procedure.
    do
      -- Create a simple feed with some categories
      create feed.make ("EiffelRSS", create {HTTP_URL}.↘
      →make ("http://eiffelrss.berlios.de"), "EiffelRSS ↘
      →news")
      feed.add_category (create {CATEGORY}.make_title ("↘
      →RSS"))
      feed.add_category (create {CATEGORY}.make_title ("↘
      →Programming"))
      feed.add_category (create {CATEGORY}.make_title ("↘
      →Eiffel"))

      -- Add a cloud to feed
      feed.create_cloud ("eiffelrss.berlios.de", 80, "/↘
      →RPC2", "xmlStorageSystem.rssPleaseNotify", "xml-↘
      →rpc")

      -- Add an image to feed
      feed.create_image (create {HTTP_URL}.make ("http↘
      →://eiffelrss.berlios.de/logo.png"), "EiffelRSS", ↘
      →create {HTTP_URL}.make ("http://eiffelrss.berlios↘
      →.de"))
```

```
			-- Add a text input field to feed
			feed.create_text_input ("Search", "Search award-winning pages", "search", create {HTTP_URL}.make ("http://eiffelrss.berlios.de/Main/SearchWiki/"))

			-- Add some simple items, use 'feed.last_added_item' or directly create an item for finer control
			feed.new_item ("Version 23 released!", create {HTTP_URL}.make ("http://eiffelrss.berlios.de/Main/News"), "Version 23 of EiffelRSS got release today. Happy syndicating!")
			feed.last_added_item.add_category (create {CATEGORY}.make_title_domain ("News", create {HTTP_URL}.make ("http://eiffelrss.berlios.de/Main/News/")))

			feed.new_item ("EiffelRSS wins award", create {HTTP_URL}.make ("http://eiffelrss.berlios.de/Main/Awards"), "EiffelRSS has been awarded by ISE as best syndication software written in Eiffel. For more info see award-winning pages: http://eiffelrss.berlios.de")
			feed.last_added_item.set_guid (create {ITEM_GUID}.make_perma_link ("http://eiffelrss.berlios.de/newsItem42"))

			-- Print feed
			io.put_string ("Sample feed:%N")
			io.put_string ("============%N%N%N")
			io.put_string (feed.to_string)
		end

feature -- Arguments

	feed: FEED
			-- Example feed

end -- class FEED_EXAMPLE
```

# Chapter 8

# Feed implementation

## 8.1 Class FEED

## 8.2 Class CHANNEL

## 8.3 Class ITEM

## 8.4 Class CATEGORY

## 8.5 Observers

# Part III

# FORMATS

# Chapter 9

# Overview

FORMATS contains classes to manage the different formats and the actual implementation of these formats.

Each format has a format, a writer and a reader object and a unique name. It also provides a feature which can detect if the format can read a certain XML document.

There is a special format called "Error" which is used whenever an error occurs.

See figure 9.1 for an overview of the cluster.

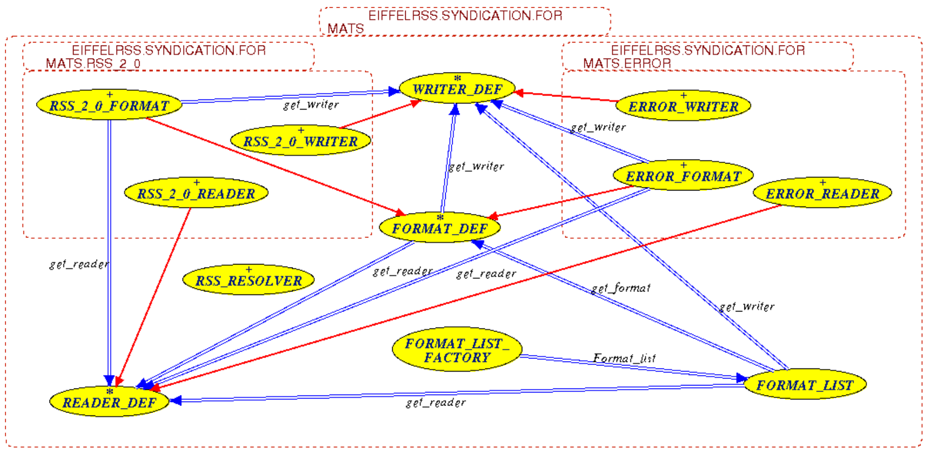

**Figure 9.1:** BON diagram of cluster FORMATS

# Chapter 10

# Management: Class FORMAT_LIST

## 10.1 Overview

`FORMAT_LIST` manages the different formats.

## 10.2 Usage

`FORMAT_LIST` uses a singleton pattern, so to actually use the format list your class has to inherit from `FORMAT_LIST_FACTORY`.

`FORMAT_LIST` inherits from `LINKED_LIST`, so all the features of `LINKED_LIST` are availabe as well.

## 10.3 Features

### 10.3.1 Initialization

**make_list**

```
make_list
  -- Create the object and add the default formats
```

### 10.3.2 Access

**get_reader**

```
get_reader (a_name: STRING): READER_DEF
  -- Get the reader object for the name 'a_name'
```

**get_writer**

```
get_writer (a_name: STRING): WRITER_DEF
  -- Get the writer object for the name 'a_name'
```

**get_format**

```
get_format (a_name: STRING): FORMAT_DEF
  -- Get the format object for the name 'a_name'
```

### 10.3.3 Detection

**detect_format**

```
detect_format (a_document: XM_DOCUMENT): STRING
  -- Get the format name for 'a_document'
```

# Chapter 11

# Format implementations

## 11.1 Addding a new format

Adding a new format to EiffelRSS? is very easy. You have to provide three objects which inherit from the deferred base classes `FORMAT_DEF`, `READER_DEF` and `WRITER_DEF`. If you only want to implement a reader or a writer, you can return `ERROR_WRITER` respectively `ERROR_READER` for the other feature.

To actually add the format to the library, you have to extend `FORMAT_LIST` with an object of the format class.

## 11.2 Base classes

### 11.2.1 FORMAT_DEF

#### get_reader

```
get_reader: READER_DEF
  -- Return a reader object
deferred
```

#### get_writer

```
get_writer: WRITER_DEF
  -- Return a writer object
deferred
```

**get_name**

```
get_name: STRING
  -- Return the format name
deferred
```

**is_of_format**

```
is_of_format (a_document: XM_DOCUMENT): BOOLEAN
  -- Is this document a feed of our type?
```

### 11.2.2 READER_DEF

**read**

```
read (a_document: XM_DOCUMENT): FEED
  -- Parse the document and return a feed
deferred
```

**get_name**

```
get_name: STRING
  -- Return a string with the format name
deferred
```

**read_or_default_element**

```
read_or_default_element (a_element: XM_ELEMENT; ↘
→default_value: STRING): STRING
  -- Read the text of 'a_element' or use 'default_value'↘
  → if 'a_element' is Void or empty
```

**read_or_default_attribute**

```
read_or_default_attribute (a_attribute: XM_ATTRIBUTE; ↘
→default_value: STRING): STRING
  -- Read the value of 'a_attribute' or use '↘
  →default_value' if 'a_element' is Void or empty
```

**valid_element_text**

```
valid_element_text (an_element: XM_ELEMENT; a_name: ↘
→STRING): BOOLEAN
  -- Has the subelement 'a_name' of 'an_element' text?
```

**read_date**

```
read_date (a_string: STRING): DATE_TIME
  -- Convert an RFC 822 date string to a DATE_TIME ↘
  →object
```

### 11.2.3 WRITER_DEF

**get_name**

```
get_name: STRING
  -- Return a string with the format name
deferred
```

**writer**

```
write (a_feed: FEED): XM_DOCUMENT
  -- Export 'a_feed' into an xml document
deferred
```

## 11.3 Built-in formats

### 11.3.1 RSS 2.0

`RSS_2_0_FORMAT` is an example implementation of the RSS 2.0 standard. It reads almost all the possible data and has a very basic writer.

### 11.3.2 Error

`ERROR_FORMAT` is a special format which is used whenever an error occurs. This removes a lot of sources of errors because the library can ensure that the reader and writer objects are never `Void`.

`ERROR_READER` returns a generated feed which has one item with the error message as description.