

# **GStreamer Plugin Writer's Guide**

**Richard John Boulton**

**Erik Walthinsen**

## **GStreamer Plugin Writer's Guide**

by Richard John Boulton and Erik Walthinsen

This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>)  
(<http://www.opencontent.org/openpub/>)

# Table of Contents

<b>I. Introduction .....</b>	<b>4</b>
1. Do I care? .....	5
2. Preliminary reading .....	6
<b>II. Basic concepts.....</b>	<b>7</b>
3. Plugins .....	8
4. Elements.....	9
5. Buffers.....	10
6. Scheduling.....	11
7. Chain vs Loop Elements .....	12
8. Typing and Properties .....	13
9. Metadata.....	14
<b>III. Building our first plugin .....</b>	<b>15</b>
10. Constructing the boilerplate .....	16
10.1. Doing it the hard way with GstObject .....	16
10.2. Doing it the easy way with FilterFactory .....	17
11. An identity filter.....	18
11.1. Building an object with pads.....	18
11.2. Attaching functions .....	18
11.3. The chain function .....	18
12. The plugin_init function .....	19
12.1. Registering the types.....	19
12.2. Registering the filter.....	19
12.3. Having multiple filters in a single plugin.....	19
<b>IV. Building a simple test application .....</b>	<b>20</b>
<b>V. Loop-based Elements.....</b>	<b>21</b>
<b>VI. Types and Properties .....</b>	<b>22</b>
<b>VII. Buffers and Metadata .....</b>	<b>23</b>
<b>VIII. Sources and Sinks .....</b>	<b>24</b>
<b>IX. State management .....</b>	<b>25</b>
<b>X. Checklist.....</b>	<b>26</b>

# I. Introduction

## Table of Contents

<b>1. Do I care? .....</b>	<b>5</b>
<b>2. Preliminary reading .....</b>	<b>6</b>

GStreamer is a framework for creating streaming media applications. It is extremely powerful and versatile, and this versatility stems in part from its modularity, and its ability to incorporate new modules seamlessly into its framework. This document describes how to extend the capabilities of GStreamer by creating new plugins.

It first describes the concepts required and the ways in which GStreamer can be extended. It then goes through a worked example of how to write a simple filter (for data processing), and how to test and debug it. More advanced concepts are then introduced, with worked examples of each. Next, writing source and sink elements (for performing input and output) is discussed. Finally, checklists of things to be sure to do when extending GStreamer are presented.

# Chapter 1. Do I care?

This guide explains how to write new modules for GStreamer. It is relevant to:

- 

Anyone who wants to add support for new input and output devices, often called sources and sinks. For example, adding the ability to write to a new video output system could be done by writing an appropriate sink plugin.

- 

Anyone who wants to add support for new ways of processing data in GStreamer, often called filters. For example, a new data format converter could be created.

- 

Anyone who wants to extend GStreamer in any way: you need to have an understanding of how the plugin system works before you can understand the constraints it places on the rest of the code. And you might be surprised at how much can be done with plugins.

This guide is not relevant to you if you only want to use the existing functionality of GStreamer, or use an application which uses GStreamer. You lot can go away. Shoo... (You might find the *GStreamer Application Development Manual* helpful though.)

## Chapter 2. Preliminary reading

The reader should be familiar with the basic workings of GStreamer. For a gentle introduction to GStreamer, you may wish to read the *GStreamer Application Development Manual*. Since GStreamer adheres to the GTK+ programming model, the reader is also assumed to understand the basics of GTK+.

# II. Basic concepts

## Table of Contents

<b>3. Plugins.....</b>	<b>8</b>
<b>4. Elements.....</b>	<b>9</b>
<b>5. Buffers.....</b>	<b>10</b>
<b>6. Scheduling.....</b>	<b>11</b>
<b>7. Chain vs Loop Elements.....</b>	<b>12</b>
<b>8. Typing and Properties .....</b>	<b>13</b>
<b>9. Metadata .....</b>	<b>14</b>

This section introduces the basic concepts required to understand the issues involved in extending GStreamer

# Chapter 3. Plugins

Extensions to GStreamer can be made using a plugin mechanism. This is used extensively in GStreamer even if only the standard package is being used: a few very basic functions reside in the core library, and all others are in a standard set of plugins.

Plugins are only loaded when needed: a plugin registry is used to store the details of the plugins so that it is not necessary to load all plugins to determine which are needed. This registry needs to be updated when a new plugin is added to the system: see the *gststreamer-register* utility and the documentation in the *GStreamer Application Development Manual* for more details.

User extensions to GStreamer can be installed in the main plugin directory, and will immediately be available for use in applications. *gststreamer-register* should be run to update the repository: but the system will work correctly even if it hasn't been - it will just load the correct plugin faster.

User specific plugin directories and registries will be available in future versions of GStreamer.

# Chapter 4. Elements

Elements are at the core of GStreamer. Without elements, GStreamer is just a bunch of pipe fittings with nothing to connect. A large number of elements (filters, sources and sinks) ship with GStreamer, but extra elements can also be written.

An element may be constructed in several different ways, but all must conform to the same basic rules. A simple filter may be built with the FilterFactory, where the only code that need be written is the actual filter code. A more complex filter, or a source or sink, will need to be written out fully for complete access to the features and performance possible with GStreamer.

The implementation of a new element will be contained in a plugin: a single plugin may contain the implementation of several elements, or just a single one.

# Chapter 5. Buffers

# Chapter 6. Scheduling

# **Chapter 7. Chain vs Loop Elements**

# Chapter 8. Typing and Properties

# Chapter 9. Metadata

# III. Building our first plugin

## Table of Contents

<b>10. Constructing the boilerplate .....</b>	<b>16</b>
<b>11. An identity filter .....</b>	<b>18</b>
<b>12. The <code>plugin_init</code> function .....</b>	<b>19</b>

We are now have the neccessary concepts to build our first plugin. We are going to build an element which has a single input pad and a single output pad, and simply passes anything it reads on the input pad through and out on the output pad. We will also see where we could add code to convert this plugin into something more useful.

The example code used in this section can be found in `examples/plugins/`

# Chapter 10. Constructing the boilerplate

The first thing to do when making a new element is to specify some basic details about it: what its name is, who wrote it, what version number it is, etc. We also need to define an object to represent the element and to store the data the element needs. I shall refer to these details collectively as the *boilerplate*.

## 10.1. Doing it the hard way with GstObject

The standard way of defining the boilerplate is simply to write some code, and fill in some structures. The easiest way to do this is to copy an example and modify according to your needs.

First we will examine the code you would be likely to place in a header file (although since the interface to the code is entirely defined by the plugging system, and doesn't depend on reading a header file, this is not crucial.) The code here can be found in `examples/plugins/example.h`

```
/* Definition of structure storing data for this element. */
typedef struct _GstExample GstExample;
struct _GstExample {
    GstElement element;

    GstPad *sinkpad,*srcpad;

    gint8 active;
};

/* Standard definition defining a class for this element. */
typedef struct _GstExampleClass GstExampleClass;
struct _GstExampleClass {
    GstElementClass parent_class;
};
```

```

/* Standard macros for defining types for this element. */
#define GST_TYPE_EXAMPLE \
    (gst_example_get_type())
#define GST_EXAMPLE(obj) \
    (GTK_CHECK_CAST((obj), GST_TYPE_EXAMPLE, GstExample))
#define GST_EXAMPLE_CLASS(klass) \
    (GTK_CHECK_CLASS_CAST((klass), GST_TYPE_EXAMPLE, GstExample))
#define GST_IS_EXAMPLE(obj) \
    (GTK_CHECK_TYPE((obj), GST_TYPE_EXAMPLE))
#define GST_IS_EXAMPLE_CLASS(klass) \
    (GTK_CHECK_CLASS_TYPE((klass), GST_TYPE_EXAMPLE))

/* Standard function returning type information. */
GType gst_example_get_type(void);

```

## 10.2. Doing it the easy way with FilterFactory

A plan for the future is to create a FilterFactory, to make the process of making a new filter a simple process of specifying a few details, and writing a small amount of code to perform the actual data processing.

Unfortunately, this hasn't yet been implemented. It is also likely that when it is, it will not be possible to cover all the possibilities available by writing the boilerplate yourself, so some plugins will always need to be manually registered.

# **Chapter 11. An identity filter**

## **11.1. Building an object with pads**

## **11.2. Attaching functions**

## **11.3. The chain function**

# **Chapter 12. The plugin\_init function**

## **12.1. Registering the types**

## **12.2. Registering the filter**

## **12.3. Having multiple filters in a single plugin**

## **IV. Building a simple test application**

Initialization Instantiating the plugins (NOTE: we really should have a debugging Sink)  
Connecting them Running the pipeline

## **V. Loop-based Elements**

How scheduling works, aka pushing and pulling How a loopfunc works, aka pulling and pushing Adding a second output Identity is now a tee Modifying the test application

## **VI. Types and Properties**

Building a simple format for testing A simple MIME type Type properties Typefind functions  
and autoplugging

## **VII. Buffers and Metadata**

Anatomy of a Buffer Refcounts and mutability Metadata How Properties work efficiently  
Metadata mutability (FIXME: this is an unsolved problem)

## **VIII. Sources and Sinks**

Writing a source Pull vs loop based Region pulling (NOTE: somewhere explain how filters use this) Writing a sink Gee, that was easy

## **IX. State management**

What are states? Managing filter state

## **X. Checklist**

