Xaraya
Request for Comments: 0041
Category: Best Current Practice

M.R. van der Boom
F. Besler
Xaraya Development Group
February 2003

Event handling system

Status of this Memo

This document specifies a Xaraya Best Current Practices for the Xaraya Community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This RFC describes the event messaging system used by Xaraya to date. The different core packages may register events and trigger them at a certain point in the system load process. The modules may define event handlers to react to these events.

Table of Contents

1 Introduction **3**

2 Registering of events **4**

3 Trigger them **5**

4 Event handler **6**

5 Inner workings **7**

6 Revision history **8**

1. Introduction

In this RFC the mechanisms will be described which are currently used in the event messaging system of Xaraya.

The event system is used by core packages of Xaraya to trigger certain events in the system. These events are dispatched to the active modules which can define special functions which can then act on those events.

This system co-exists with the hooks system. They are alike but the main reasons to have a separate event-system next to the hook system are:

1. The hooks live in *module-space* while we need signalling from the core as well
2. The events are *static*, they are guaranteed to happen, while hooks are dynamic and can be disabled (or rather, not configured) by the site administrator.

2. Registering of events

Some core packages can define events, also known as registering them, by calling the function.

```
xarEvt_registerEvent($eventName)
```

in its initialization function. This function registers the name of the event in a global variable. The only place that this function may be called is in a core package initialization part. This can be checked with bk -r grep xarEvt_registerEvent.

Not all core packages can register events, for example the exception system cannot. The reason for this is that the exception system cannot depend on anything but the basics in Xaraya.

The naming convention says that the event name starts with the core package name that registers it (the owner), beginning with a capital letter. The specific name then follows directly, also beginning with a capital letter. No spaces or special characters are allowed. Example:

```
xarEvt_registerEvent('ServerRequest');
```

3. Trigger them

At certain points in the system load process the registered events can be triggered by calling the function

```
xarEvt_trigger($eventName, $value)
```

with $value being an optional variable that can be passed to the event handlers. Example:

```
xarEvt_trigger('ServerRequest');
```

This function always returns. It might be the case that event handlers do not exist or that event handlers do something funny. The system relies on the exception handling system to catch these situations. The xarEvt_Trigger function neither sets directly exceptions nor handles them from its callees.

4. Event handler

Each module, in turn, can implement an event handler function, that will be executed as soon as the corresponding event is being triggered by the core. The function must be situated in the file xareventapi.php of the module and must adhere the following naming convention:

```
function {$modName}_eventapi_On$EventName($arg)
{
  // event handler code, e.g. db interaction, ...
}
```

Example:

```
function sniffer_eventapi_OnSessionCreate($arg)
{
        // browsersniffing
    }
```

5. Inner workings

As soon as an event is being triggered somewhere in the core, the events system looks up in each module whether it implements the corresponding event handler. If so, the function is passed the optional variable from the trigger call and is executed immediately. There is no guarantee for call order in case there is more than one module that implements a certain event handler.

6. Revision history

2002-02-27 : Version 0.2

2003-02-26 : Version 0.1

2003-02-23 : Created