



Pedagogically sustained Adaptive Learning through  
the Exploitation of Tacit and Explicit Knowledge

# **Palette Widget Format Specification**

**January 29<sup>th</sup> 2009**

**Jérôme Bogaerts**  
**Laurent Haan**  
**Stéphane Sire**  
**Alain Vagner**

# Index

1	Introduction .....	3
2	License .....	4
3	PALETTE Widget Format Specification.....	5
3.1	Introduction.....	5
3.2	Widget Packaging .....	5
3.2.1	File format .....	5
3.2.2	Widget files.....	5
3.2.3	Widget Folder Structure .....	5
3.2.4	Widget Configuration File: config.xml.....	6
3.3	Widget Scripting Interfaces .....	9
3.3.1	The widget Object for Local Widgets.....	9
3.4	User Preferences Access for Remote Widgets .....	13
3.5	Widget descriptor schema .....	13
3.5.1	Manifest.xsd.....	13
3.5.2	Palette.xsd.....	14
3.6	PALETTE Widgets 1.0 Tutorial .....	16
3.6.1	Introduction.....	16
3.6.2	Widget Configuration File: config.xml.....	16
3.6.3	Widget Index File.....	18
4	References .....	23

# 1 Introduction

The Palette Widget Specification Format was designed in the frame of the Palette project (URL: PAL). Palette, for Pedagogically sustained Adaptive Learning through the Exploitation of Tacit and Explicit Knowledge, aims at facilitating and augmenting individual and organisational learning in Communities of Practice (CoPs). Towards this aim, an interoperable and extensible set of innovative services as well as a set of specific scenarios of use have been designed, implemented and thoroughly validated in CoPs of diverse contexts.

The Palette project provides a wide range of services focusing mainly on Knowledge Management, Learning, and Information sharing. Lots of different stand-alone and web applications were developed, composing a huge heterogeneous information system, using a resource oriented architecture sharing XML-based file as a homogenous pivot format. However, the project needed for a unique homogenous entry point for end-users at the view level. A Widget Portal was thus developed to provide users a way to have a global point of view on their Community of Practice. In this way, we developed this Palette Widget Specification Format to foster the adoption of the Widget Portal paradigm within Palette Technical Partners, and help them to develop their own widgets.

In this situation, we decided to create our own Widget Portal, known as myWiWall, implementing successfully the Palette Widget Specification Format. You will find further information about this implementation on the *CRP Henri Tudor contributions to the Palette project* website (URL: PALU).

Palette is an Integrated Project supported by the IST programme of the European Commission (DG Information Society and Media, project no. 028038). We thank warmly the Palette FP6 project for its financing.

## 2 License

This document is licensed under the **Creative Commons Attribution-Share Alike 3.0 Unported (CC-BY-SA 3.0)**. Please visit <http://creativecommons.org/licenses/by-sa/3.0/> for more information about this license.

© Centre de Recherche Public Henri Tudor – Ecole Polytechnique Fédérale de Lausanne, 2009

## 3 PALETTE Widget Format Specification

### 3.1 Introduction

Widgets, as they are used within the PALETTE portal, are small Web applications for displaying and updating remote data and serve as a common entry point to the end user. The PALETTE Widget format is inspired by the W3C Widgets 1.0 Draft (URL: W10) with small implementation-specific adaptations and extensions, while being fully compatible to the original Widgets 1.0 specification.

The PALETTE widgets specification defines two different types of widgets: local and remote widgets. A local widget is a bundled archive of files that is deployed within the portal. Remote widgets are stored on a remote server and only registered within the portal by supplying their manifest, which contains the URI where the widget can be found. They can be written in any language as long as they generate valid HTML. Differences between local and remote widgets exist in the packaging, some elements within the manifest *config.xml* as well as the Widget Scripting Interfaces.

### 3.2 Widget Packaging

#### 3.2.1 File format

The file format only applies to local widgets, as they are uploaded and deployed within the portal. Remote widgets can use whatever file format their implementation language requires, as long as they can be referenced by an URI.

Local widgets are a bundled archive of files, as specified by the Widgets 1.0 Draft File Format (URL: W10F).

#### 3.2.2 Widget files

Every widget *must* define the following two files:

The *config.xml* file:

This manifest file contains information necessary to initialise the widget. It always contains information about the widget's name, identity and geometry and may optionally contain more information about the widget, such as the widget description, author information and an icon reference.

An index file:

This is the main document of the widget, which is displayed in the portal and whose main properties are established by the *config.xml* file. For local widgets, the index file *must* be called *index.html*. For remote widgets, this *must* be a valid index file. Since remote widgets are hosted on remote servers, the validity of the index file name depends on the web server configuration. The index document can reference external content and *should* produce valid HTML or XHTML markup.

#### 3.2.3 Widget Folder Structure

The widget folder structure only applies to local widgets. Remote widgets can use any folder structure as long as the widget can be referenced by an URI pointing to its folder.

The *config.xml* and *index.html* *must* be at the root of the .zip file, with any associated resources, such as scripts and images, in the same directory or subdirectories.

### 3.2.4 Widget Configuration File: config.xml

Necessary information to run and display a widget within the portal are stored in a file named *config.xml*. The *config.xml* file is an XML document.

Since the PALETTE widget specification is an extension of the W3C Widgets 1.0 specification, two different namespaces are used to distinguish between the two types of elements. These namespaces have to be defined in the root *widget* element: the default W3C namespace *must* be <http://www.w3.org/TR/widgets/> and the PALETTE namespace *must* be <http://palette.ercim.org/ns/>

A minimal *config.xml* looks like the following, giving the widget a name, an initial viewport of 300×300 pixels and an id:

```
<widget xmlns="http://www.w3.org/TR/widgets/" xmlns:palette=http://palette.ercim.org/ns/
  id="helloWorldWidget"
  width="300"
  height="300">
  <title>Hello World!</title>
</widget>
```

For the *config.xml* file, the same structural restrictions apply as for the Widgets 1.0 Draft, specified in chapter 3 (URL: W10C).

To validate the *config.xml* file, two XML Schema files are provided with this document.

#### The widget Element

The *widget* element is the root element of the *config.xml* file and *must* be present. It *should* contain, in any order, exactly one title, and optionally one of each author, description and icon. Please note that the widget id attribute is mandatory (contrary to Widgets 1.0 Draft) and is used to identify the widget internally in the PALETTE portal.

#### The id attribute

The *id* attribute *must* be present in *config.xml* and bound to the *widget* element. The identifier should be a valid XML Schema ID (URL: XSDID).

#### The width and height attributes

The *width* and *height* attributes *must* be present in *config.xml* as children of the *widget* element. After stripping of any leading/trailing white space, the value of this element *must* be interpretable as a string representation of an integer, containing only the characters [0-9].

Please note that, while these integers give the initial width and height of the viewport of the widget, measured in CSS pixels (see section 4.3.2 of URL: CSS), only the *height* integer is directly respected by the portal, while the width of the widget is defined by the portal layout. The *width* integer serves only for compatibility with the W3C specification and might be used in the future to display the widget as a Desktop widget (URL: WD).

#### The title Element

The *title* element *must* be present in *config.xml* as a child of the *widget* element. It *should* contain a string whose purpose is to provide a human-readable title for the widget that can be used for example in application menus and similar contexts.

### The author, description and icon Elements

These elements respect the specification given by the Widgets 1.0 Draft. Please refer to chapter 3 of the Widgets 1.0 Draft for further details.

### The `widget_type` Element

The `widget_type` element has been added to the Widgets 1.0 Working Draft and *must* be specified in the palette namespace.

The `widget_type` is an optional element of the `widget` element and the absence of the `widget_type` element means that the widget is a local widget. It *should* contain a string whose only allowed values are either *local* or *remote*.

```
<palette:widget_type>local</palette:widget_type>
```

### The `widget_location` Element

The `widget_location` element applies only to remote widgets. It has been added to the Widgets 1.0 Working Draft and *must* be specified in the palette namespace.

If the `widget_type` element value is *remote*, the `widget_location` element *must* be present in `config.xml` as a child of the `widget` element. If the `widget_type` element value is *local*, this element is ignored. The value of this element is interpreted as a syntactically valid URI pointing to the folder containing the remote widget index file. If no trailing slash is present, the

```
<palette:widget_location>http://path.to/my/remote/widget/</palette:widget_location>
```

portal will automatically add one.

### The `alternate_url` Element

The `alternate_url` element has been added to the Widgets 1.0 Working Draft and *must* be specified in the palette namespace.

The `alternate_url` is an optional element of the `widget` element and allows to specify an alternate view of the resource, the widget is about. In the case of remote widgets, `alternate_url` can be used to provide an URL to the entire application of which the widget shows only one functionality.

### The `preferences` Element

The `preferences` element has been added to the Widgets 1.0 Working Draft and *must* be specified in the palette namespace.

The `preferences` element allows to specify customisable user preferences that are stored in the PALETTE Web portal and exposed to the widgets through the Widget Scripting Interfaces. The `preferences` element contains any number of `preference` elements.

### The `preference` Element

The `preference` element has been added to the Widgets 1.0 Working Draft and *must* be specified in the palette namespace.

If present, this element *must* be a child of the `preferences` element. The following table lists the `preference` attributes:

<b>name</b>	Required "symbolic" name of the user preference; displayed to the user during editing if no <code>display_name</code> is defined. Must contain only letters, number and underscores, <i>i.e.</i> the regular expression <code>^[a-zA-Z0-9_]+\$</code> must be unique.
<b>display_name</b>	Optional string to display alongside the user preferences in the edit window. Must be unique.
<b>datatype</b>	Optional string that indicates the data type of this attribute. Can be string, bool, number, hidden or enumeration. The default is string.
<b>default_value</b>	Optional string that indicates a user preference's default value.

```
<palette:preferences>
  <palette:preference name="name" display_name="First name" datatype="string"/>
  <palette:preference name="age" display_name="Age" datatype="number"/>
</palette:preferences>
```

### The enumeration Element

The *enumeration* element has been added to the Widgets 1.0 Working Draft and *must* be specified in the palette namespace.

If present, this element *must* be a child of the *preference* element. This element is ignored if the attribute *datatype* of the parent *preference* element is not of value *enumeration*. The following table lists the *enumeration* attributes:

<b>value</b>	Required "symbolic" value of the user preference; displayed to the user during editing if no <code>display_value</code> is defined. Must contain only letters, number and underscores, <i>i.e.</i> the regular expression <code>^[a-zA-Z0-9_]+\$</code> must be unique.
<b>display_value</b>	Optional string to display alongside the user preferences in the edit window. Must be unique.

The *enumeration* data type is presented in the user interface as a menu of choices. The content of the menu is specified using the enumeration elements.

```
<palette:preferences>
  <palette:preference name="lang" display_name="Language" datatype="enumeration"
  default_value="fr">
    <palette:enumeration value="fr" display_value="French"/>
    <palette:enumeration value="de" display_value="German"/>
    <palette:enumeration value="en" display_value="English"/>
  </palette:preference>
</palette:preferences>
```

## The authentication Element

The *authentication* element has been added to the Widgets 1.0 Working Draft and *should* be specified in the palette namespace, but is not mandatory. If present, this element *must* a child of the *widget* element. The value of the *authentication* element must be a string with a value of 'enabled' or 'disabled'.

```
<palette:authentication>enabled</palette:authentication>
```

If the *authentication* element is not set in the *config.xml* file, the default behaviour is to set the widget authentication value to 'disabled'.

## 3.3 Widget Scripting Interfaces

### 3.3.1 The widget Object for Local Widgets

The purpose of the *widget* object is to expose functionality to widgets that are not available outside of the PALETTE portal. The *widget* object is accessible through JavaScript, but only for local widgets.

#### The widget Object methods

##### method `<datatype> preferenceForKey (string key)`

The *preferenceForKey()* method takes a string argument, *key*. When called, this method *shall* return the value of the user preference whose attribute *name* corresponds to the *key* provided as argument, or null if the *key* does not exist or hasn't been specified by the user.

##### method `void setPreferenceForKey (<datatype> preference, <string> key)`

The *setPreferenceForKey()* method takes two arguments, *preference* and *key*. When called, this method shall store the value of *preference* in the preference which attribute *name* corresponds to the *key* provided as argument. If no corresponding preference has been found (the *key* doesn't correspond to a preference specified in the manifest *config.xml*), this instruction is ignored.

##### method `void httpGet (string URI, map params, function callback, function errorCallback)`

This method loads the remote page specified by the argument *URI* using an asynchronous HTTP GET request. The second argument *params* consists of key/value pairs that will be sent to the server. The result of the request, which is passed as the first argument to the function specified by *callback*, is intelligently parsed as either *responseXML* or *responseText*. The *errorCallback* parameter enables the developer to handle an eventual error during the *XMLHttpRequest* process. The signature of this *errorCallback* function has 3 parameters that are *xhr* (the *XMLHttpRequest* object), *status* (a string depicting the status of the request) and *thrownError* (the ultimate exception thrown when the request failed).

##### method `void httpPost (string URI, map params, function callback, function errorCallback)`

This method loads the remote page specified by the argument *URI* using an asynchronous HTTP POST request. The second argument *params* consists of key/value pairs that will be sent to the server. The result of the request, which is passed as the first argument to the function specified by *callback*, is intelligently parsed as either *responseXML* or *responseText*. The

signature of this *errorCallback* function has 3 parameters that are *xhr* (the XMLHttpRequest object), *status* (a string depicting the status of the request) and *thrownError* (the ultimate exception thrown when the request failed).

**method void *httpPut* (string *URI*, map *params*, function *callback*)**

This method loads the remote page specified by the argument *URI* using an asynchronous HTTP PUT request. The second argument *params* consists of key/value pairs that will be sent to the server. The result of the request, which is passed as the first argument to the function specified by *callback*, is intelligently parsed as either *responseXML* or *responseText*.

**method void *httpDelete*(string *URI*, map *params*, function *callback*)**

This method loads the remote page specified by the argument *URI* using an asynchronous HTTP DELETE request. The second argument *params* consists of key/value pairs that will be sent to the server. The result of the request, which is passed as the first argument to the function specified by *callback*, is intelligently parsed as either *responseXML* or *responseText*.

**method void *httpGetJSON* (string *URI*, map *params*, function *callback*, function *errorCallback*)**

This method loads the remote page specified by the argument *URI* using an asynchronous HTTP GET request. The second argument *params* consists of key/value pairs that will be sent to the server. The result of the request, which is passed as the first argument to the function specified by *callback*, is parsed as JSON into a JavaScript object. The signature of this *errorCallback* function has 3 parameters that are *xhr* (the XMLHttpRequest object), *status* (a string depicting the status of the request) and *thrownError* (the ultimate exception thrown when the request failed).

The *httpPost*, *httpGet*, *httpPut*, *httpDelete* and *httpGetJSON* methods above profit from a proxy installed within the PALETTE Web portal and are able to bypass the security restrictions imposed by the XMLHttpRequest object. This makes it possible to send a request to an URI that is not on the PALETTE Web server but can be any remote server.

**method void *setContentProxy* (string *path*)**

The *setContentProxy* method is used to specify the location of the content proxy PHP file to be used by the local development environment. The local development environment is a special environment designed to facilitate widget development without the need to deploy and test the widget within the PALETTE Web portal. It is possible to use this environment by simply including two specific files, a CSS and a JavaScript file, in the HTML code of the *index.html* file, as explained further in the Widget Tutorial. Since the proxy needs to be installed on the server the widget is executed on, it is necessary to specify the path to the PHP proxy on the local machine.

**method void *setHttpCredentials* (string *username*, string *password*)**

The *setHttpCredentials* method is used to specify the username and password for basic HTTP access authentication.

**method void *fireWidgetEvent* (string *target*, string *eventType*, Object *data*)**

The *fireWidgetEvent* method fires an event of a given *eventType* in order to transmit some *data* in the transmitted message, for a particular *target* widget alphanumeric identifier. If the *target* is set to *null*, the event will be notified to every widget instance listening for this event type.

**method void addWidgetEventListener (string eventType, function eventHandler, Array acceptedSources)**

The addWidgetEventListener method enables your widgets to listen to *eventType* events that are coming from a specified alphanumeric widget identifier list defining the *acceptedSources*. When events are fired, the function defined by *eventHandler* is executed. If the *acceptedSources* list is set to null, the widget will accept be notified of this type of event from any source widget.

The function *eventHandler* must accept an object as its first parameter. Indeed, this object will represent the message transmitted when events are notified to listeners. The object will be an Object instance with two properties:

- *eventType*: a string containing a URI that represents the type of event that was fired.
- *data*: some arbitrary useful to handle the event. The format of the transmitted data may differ between event types.

**method void removeWidgetEventListener (string eventType, string source)**

Stop to listen to a given *eventType* for a particular *source* widget identified by an alphanumeric widget identifier. If the **source** is set to *null*, the widget will completely stop to listen to this kind of events.

**method void bindWidgetToDropType (string eventType)**

Declares that a widget is a drop target for drag and drop events of a given *eventType*. This will inform the drag and drop widget engine to display a proper feedback when dragging some compatible data over the widget frame.

**method void addDragData (Element domElement, string eventType, Object data, string tooltip)**

Transforms a *domElement* of the widget tree into a draggable element that triggers a drag event of a given *eventType* with an associated *data* and that displays a *tooltip* message when dragging. The widgets that registered for the drag event of type *eventType* with the *bindWidgetToDropType* method will be notified if the user drops the widget element onto them.

**method void openURL (string url)**

Opens the resource at the location specified by *url* in a new window.

**method void show ( )**

Shows the viewstate of the widget.

**method void hide ( )**

Hides the viewstat of the widget.

**method void getAttention ( )**

Gets the attention of the user. This method can be implemented as a graphical, textual, etc, notification near the widget.

**method void enableAuthentication ( )**

By invoking this method, the widget will be forced to use the Widget Authentication Mechanism, even if it was not defined in the widget's XML manifest.

### The `onLoad()` function for Local Widgets

The `onload()` function replaces the Intrinsic event `onload` specified by the HTML 4 Scripts document (URL: H4S). This event occurs when the user agent finishes loading a window or all frames within a frameset and is commonly used to execute JavaScript as soon as the page has finished loading. Since the PALETTE portal needs to initialize the `widget` object first, we discourage the usage of the `onload` event, since we cannot guarantee that the `widget` object has been fully loaded. The `onload` event should be replaced by the `onLoad()` function, if the widget needs to execute a script when it has finished loading.

The `onLoad()` function is optional and if present, is called by the PALETTE portal as soon as the widget has been fully initialised. To guarantee that the HTML DOM is ready and the `widget` object accessible, the widget logic should be executed from within the `onLoad()` function.

```
<script type="text/javascript">
function onLoad()
{
    /* put your widget logic here */
}
</script>
```

### The Widget object attributes

#### **string Id**

The unique identifier of the widget.

#### **string Title**

The current title of the widget.

#### **integer defaultTitle**

The default title of the widget.

#### **integer defaultHeight**

The default height of the widget's viewstate.

#### **integer defaultWidth**

The default width of the widget's viewstate.

#### **integer height**

The current height of the widget's viewstate.

#### **integer width**

The current width of the widget's viewstate.

#### **boolean useAuthentication**

Enables the widget developer to know if the widget is presently using the Widget Authentication Mechanism or not.

**boolean minimized**

Enables the widget developer to know if the widget's viewstate is presently minimized (hidden) or not.

**boolean maximized**

Enables the widget developer to know if the widget's viewstate is presently maximized (shown) or not.

**string authorName**

The name of the widget author.

**string authorEmail**

The email address of the widget author

**string name**

The name of the widget.

**string description**

The description of the widget.

**3.4 User Preferences Access for Remote Widgets**

Remote widgets access the user preferences through the GET parameters specified in the widget URL. For each user preference, for which a value has been specified or a default value is known, a (key, value) pair is generated and appended to the URL.

Apart from the widget edit window in the PALETTE portal, remote widgets have no possibility to change the value of the user preference through the Widget Scripting Interface.

**3.5 Widget descriptor schema****3.5.1 Manifest.xsd**

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="http://www.w3.org/TR/widgets/"
  xmlns="http://www.w3.org/TR/widgets/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:palette="http://palette.ercim.org/ns/"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
<xs:import namespace="http://palette.ercim.org/ns/"
  schemaLocation="palette.xsd"/>
<xs:element name="widget">
  <xs:complexType>
    <xs:all>
      <xs:element ref="title" maxOccurs="1"/>
      <xs:element ref="description" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="icon" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="access" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="author" minOccurs="0"/>
      <xs:element ref="license" minOccurs="0"/>
      <xs:element ref="palette:widget_type" minOccurs="0" />
    </xs:all>
  </xs:complexType>
</xs:element>
```

```

        <xs:element ref="palette:widget_location" minOccurs="0"/>
        <xs:element ref="palette:alternate_url" minOccurs="0"/>
        <xs:element ref="palette:preferences" minOccurs="0"/>
    </xs:all>
    <xs:attribute name="id" type="xs:ID" use="required"/>
    <xs:attribute name="version" type="xs:string" use="optional"/>
    <xs:attribute name="height" type="xs:positiveInteger" use="optional"/>
    <xs:attribute name="width" type="xs:positiveInteger" use="optional"/>
    <xs:attribute name="start" type="xs:string" use="optional"/>
</xs:complexType>
</xs:element>

<xs:element name="title" type="xs:string"/>
<xs:element name="description" type="xs:string"/>

<xs:element name="icon">
    <xs:complexType>
        <xs:attribute name="src" type="xs:anyURI"/>
    </xs:complexType>
</xs:element>

</xs:complexType>
</xs:element>

<xs:element name="access">
    <xs:complexType>
        <xs:attribute name="network" type="xs:boolean"/>
        <xs:attribute name="plugins" type="xs:boolean"/>
    </xs:complexType>
</xs:element>

<xs:element name="author">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="url" type="xs:anyURI"/>
                <xs:attribute name="email" type="xs:string"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

<xs:element name="license">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="href" type="xs:anyURI"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>

</xs:schema>

```

### 3.5.2 Palette.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://palette.ercim.org/ns/"
xmlns="http://palette.ercim.org/ns/">

<xs:element name="widget_type">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="local"/>
      <xs:enumeration value="remote"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="widget_location" type="xs:anyURI"/>

<xs:element name="alternate_url" type="xs:anyURI"/>

<xs:element name="preferences">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="preference"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="preference">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="enumeration"/>
    </xs:choice>
    <xs:attribute name="name" type="identifier" use="required"/>
    <xs:attribute name="display_name" type="xs:string" use="optional"/>
    <xs:attribute name="datatype" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="string" />
          <xs:enumeration value="bool" />
          <xs:enumeration value="number" />
          <xs:enumeration value="hidden" />
          <xs:enumeration value="enumeration" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="default_value" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:element name="enumeration">
  <xs:complexType>
    <xs:attribute name="value" type="identifier" use="required"/>
    <xs:attribute name="display_value" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="identifier">
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z0-9_]+"/>
  </xs:restriction>
</xs:simpleType>

</xs:schema>

```

## 3.6 PALETTE Widgets 1.0 Tutorial

### 3.6.1 Introduction

The PALETTE Widgets 1.0 specification has been designed with the clear goal in mind to make the creation of PALETTE Widgets as simple as possible. Only a few simple steps are necessary to make a Widget compatible with the requirements, after which you can entirely focus on the content of your Widget. This tutorial will guide you through the creation of a manifest *config.xml* and explain the few guidelines you need to follow to develop a PALETTE Widget, either local or remote.

### 3.6.2 Widget Configuration File: config.xml

#### Basic Configuration File Structure

Within a new directory, create a new text file called *config.xml*. This file contains all the information the PALETTE portal needs to deploy and run your widget as well as to save the user preferences within the portal. As you can see in the PALETTE Widgets 1.0 specification, only a few elements are mandatory and a minimal *config.xml* looks like the following:

```
<widget widget xmlns="http://www.w3.org/TR/widgets/"
  xmlns:palette=http://palette.ercim.org/ns/
  id="helloWorldWidget"
  height="300"
  width="300">
  <title>Local Hello World</title>
</widget>
```

If you are using an XML editor capable of validating an XML file against an XML Schema, you can modify the *widget* element to include the location of your schema:

```
<widget xmlns=http://www.w3.org/TR/widgets/
  xmlns:palette="http://palette.ercim.org/ns/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/TR/widgets/manifest.xsd"
  id="helloWorldWidget"
  height="300"
  width="300">
  <title>Local Hello World</title>
</widget>
```

For the XML Schema validation to work, both the *manifest.xsd* and the *palette.xsd* should be in the same directory as the *config.xml* file. Using all of the elements defined by the Widgets 1.0 Draft, a more complete manifest could look like the following:

```
<widget xmlns="http://www.w3.org/TR/widgets/"
  xmlns:palette="http://palette.ercim.org/ns/"
```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/TR/widgets/ manifest.xsd"
id="helloWorldWidget"
height="300"
width="300">
<title>Local Hello World</title>
<author url=" http://www.tudor.lu" email="haan@tudor.lu">Laurent Haan</author>
<description>A simple Hello World widget</description>
<icon src="images/icon.gif"/>
</widget>

```

### PALETTE Specific Configurations

In order to use any of the PALETTE Web portal features, we have the possibility to use the elements declared in the palette namespace, which allow us to define the type of widget we're going to create and the user settings the portal should store.

To have the PALETTE Services Portal store the name of the widget user, we first need to declare a preference:

```

<palette:preferences>
<palette:preference name="username" display_name="Username" datatype="string"
default_value="guest"/>
</palette:preferences>

```

The complete config.xml file would look like the following:

```

<widget xmlns="http://www.w3.org/TR/widgets/"
xmlns:palette="http://palette.ercim.org/ns/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/TR/widgets/ manifest.xsd"
id="helloWorldWidget"
height="300"
width="300">
<title>Local Hello World</title>
<author url=" http://www.tudor.lu" email="haan@tudor.lu">Laurent Haan</author>
<description>A simple Hello World widget</description>
<icon src="images/icon.gif"/>
<palette:preferences>
<palette:preference name="username" display_name="Username"
datatype="string" default_value="guest"/>
</palette:preferences>
</widget>

```

### Widget Authentication Mechanism

Another PALETTE widget feature is to enable widgets to automatically authenticate themselves and their users against PALETTE Services. This feature is named “Widget Authentication Mechanism”, and enables the portal to send request on behalf of the widget. The “Widget Authentication Mechanism” solves the problem relevant to automatic authentication of users through widgets by providing a secure identity transportation system between the portal and compliant PALETTE services.

When enabled, the portal's proxy will inject on the fly some encrypted information about the widget and the user using it in requests' headers each time a request to the relevant PALETTE Service is performed.

By decrypting the data, the PALETTE Service will be able to authenticate both the widget's user and the widget by using a symmetric encryption key shared by the portal and itself.

To enable the "Widget Authentication Mechanism" for your widget, simply add the following line into your *config.xml*, between the *widget* element tags.

```
<palette:authentication>enabled</palette:authentication>
```

If you do not want to use this feature for your widget, simply omit the *authentication* element or declare it unequivocally by using the following example.

```
<palette:authentication>disabled</palette:authentication>
```

### Widget scrollbar

You may want to enable vertical scrolling for your widget if its height may vary during execution. To do so, simply add the following line into your *config.xml*, between *widget* element tags.

```
<palette:scrollable>true</palette:scrollable>
```

If you do not want to enable vertical scrolling for your widget, omit the *scrollable* element or declare it by inserting the following line your *config.xml* file.

```
<palette:scrollable>>false</palette:scrollable>
```

### 3.6.3 Widget Index File

#### Creating a Local Widget

The local widget is the default type of PALETTE Widget. It needs to be packaged and deployed in the PALETTE Web portal but has less technical restrictions because the Widget is located locally on the server. In the case of a local widget, this file must be called *index.html* and the HTML code for our "Local Hello World" Widget looks like the following:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<HTML>
<HEAD>
<TITLE>Widget Test</TITLE>
</HEAD>
<body>
Hello <span id="username">username</span>!
</body>
```

```
</HTML>
```

To access the user preferences, we use the Widget Scripting Interface, more specifically, the *widget* object that is only available within the PALETTE Web portal. Since this object might not be available immediately after the onload event, we strongly recommend to use the `onLoad()` function as explained in the PALETTE Widget 1.0 specification. If present, this function will be called by the PALETTE Web portal as soon as the entire object has been loaded and should replace the default onload event for local widgets. The complete JavaScript code that replaces the username with the value within the widget object looks like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<HTML>
<HEAD>
<TITLE>Widget Test</TITLE>

<script type="text/javascript">
  function onLoad()
  {
    document.getElementById('username').firstChild.nodeValue =
      widget.preferenceForKey('username');
  }
</script>

</HEAD>
<body>
Hello <span id="username">username</span>!
</body>
</HTML>
```

Please note that the argument given to `preferenceForKey()` is the value given for the *name* attribute for our preference in the `config.xml` file.

### Testing a Local Widget

To facilitate development of local widgets, a special development environment is available that simulates the PALETTE Web portal. This testing environment is very easy to use and only requires two lines to be added to the local widget `index.html` file:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<HTML>
<HEAD>
<TITLE>Widget Test</TITLE>

<link rel="stylesheet" type="text/css"
href="http://www.palette.tudor.lu/widget/css/palette.css" />
<script type="text/javascript"
src="http://www.palette.tudor.lu/widget/js/palette.js"></script>

<script type="text/javascript">
```

```
function onLoad()
{
    document.getElementById('username').firstChild.nodeValue =
widget.preferenceForKey('username');
}
</script>
</HEAD>
<body>
Hello <span id="username">username</span>!
</body>
</HTML>
```

These two lines will load the most recent version of the development environment and display the local widget in much the same way as it would appear within the PALETTE Web portal. The JavaScript file includes the config.xml parser that additionally checks your config.xml for syntax errors and the *widget* object that would only be available within the portal. Since no user settings are available for the development environment, the *default\_value* of the preference will be used instead.

### Testing a Local Widget with the Content Proxy

The content proxy is a script (a PHP file) that transparently loads web pages from remote servers, without the XMLHttpRequest security restrictions. It is installed within the PALETTE Web portal and allows widgets to access content from remote servers. Since the local development environment has no such proxy, we provide you with the necessary functionalities to install and use a proxy on your local machine.

The first necessary step is to download and copy the contentProxy.php file to your local machine and make it accessible by your local web server. Since the script is a PHP file, you need a webserver capable of interpreting PHP files as well. Please notice that the PHP proxy requires the CURL package to be installed and properly configured.

Secondly, you should use the setContentProxy method of the widget object to set the path to the proxy file. Afterwards, you can start using the get and post methods of the widget object as usual.

An example of a widget using a local content proxy to read the content of a text file can found below:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<title>Widget Object Test</title>
<link rel="stylesheet" type="text/css"
href="http://www.palette.tudor.lu/widget/css/palette.css" />
<script type="text/javascript" src="http://localhost/widget/js/palette.js"></script>
<script type="text/javascript">
function onLoad(){

widget.setContentProxy('./contentProxy.php');

widget.httpGet('a.txt', null, function(data){
    document.getElementById('user').appendChild(document.createTextNode(data));
});
}
</script>
</head>
<body>
Bonjour <span id="user"></span>.
</body>
</html>

```

### Creating a Remote Widget

Contrary to local widgets, remote widgets are hosted on a remote server. They can be written using any available technology, as long as the output is valid HTML or XHTML and they are valid index files on the remote server. In case of remote widgets, the user preferences are sent by GET parameters, which don't require any particular development environment to simulate. Before we are able to create a remote widget, we first need to modify the config.xml file accordingly:

```

<widget xmlns="http://www.w3.org/TR/widgets/"
  < xmlns:palette="http://palette.ercim.org/ns/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/TR/widgets/ manifest.xsd"
  id="helloWorldWidget"
  height="300"
  width="300">
  <title>Remote Hello World</title>
  <author url=" http://www.tudor.lu" email="haan@tudor.lu">Laurent
Haan</author>
  <description>A simple Hello World widget</description>
  <icon src="images/icon.gif"/>
  <palette:widget_type>remote</palette:widget_type>

  <palette:widget_location>http://path.to/my/widget/folder</palette:widget_location>
  <palette:preferences>
    <palette:preference name="username" display_name="Username"
datatype="string" default_value="guest"/>
  </palette:preferences>

```

```
</widget>
```

If we want to create a remote Hello World widget in PHP, all that is left to do is create an "index.php" file at the address specified by the *widget\_location* element and write the corresponding PHP code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<HTML>
<HEAD>
<TITLE>Widget Test</TITLE>
</HEAD>
<body>
Hello <span id="username"><?php echo $_GET['username']; ?></span>!
</body>
</HTML>
```

## 4 References

PAL	<a href="http://palette.ercim.org">http://palette.ercim.org</a>
PALU	<a href="http://palette.tudor.lu/content/index.php?page=palette-services-portal">http://palette.tudor.lu/content/index.php?page=palette-services-portal</a>
W10	<a href="http://www.w3.org/TR/2006/WD-widgets-20061109/">http://www.w3.org/TR/2006/WD-widgets-20061109/</a>
W10F	<a href="http://www.w3.org/TR/2006/WD-widgets-20061109/#file-format">http://www.w3.org/TR/2006/WD-widgets-20061109/#file-format</a>
W10C	<a href="http://www.w3.org/TR/2006/WD-widgets-20061109/#configuration">http://www.w3.org/TR/2006/WD-widgets-20061109/#configuration</a>
XSDID	<a href="http://www.w3.org/TR/xmlschema-2/#ID">http://www.w3.org/TR/xmlschema-2/#ID</a>
CSS	<a href="http://www.w3.org/TR/CSS2/syntax.html#length-units">http://www.w3.org/TR/CSS2/syntax.html#length-units</a>
WD	<a href="http://en.wikipedia.org/wiki/Widget_engine#Desktop_widgets">http://en.wikipedia.org/wiki/Widget_engine#Desktop_widgets</a>
H4S	<a href="http://www.w3.org/TR/REC-html40/interact/scripts.html">http://www.w3.org/TR/REC-html40/interact/scripts.html</a>