JKO Content Development Kit

1. Package Structure

Learning content to be accessed via the JKO mobile application is ‘packaged’ by content authors into files that contain:

¾ HTML and CSS documents

¾ Rich media resources

¾ Metadata files

A package contains all of the resources required by a piece of learning content and, once downloaded by the application, remains available to the user even when offline.

Each package is distributed as a ZIP file. There are no restrictions on how learning content is structured or the use of folders within the file other than:

¾ Two files ‘package.xml’ and ‘package.version’ must exist in the root folder of the file.

¾ HTML and CSS documents must use relative references between files

The file ‘package.xml’ describes the structure of the package, its content and any application menus. The file ‘package.version’ is used for version control. They are described later in this document.

Content authors may include any types of file they wish – including HTML, CSS, JavaScript, video and electronic book formats – within a package provided it is a supported resource-type (resources are described later in this document) or the content author has included a display mechanism.

At this release a maximum package size of 50MB is recommended, although no size limit is enforced by the application. For larger courses, a method is provided via the Handset API so content developers can segment courses into multiple packages and provide links between them.

Other than the inclusion of the mandatory files, there are no restrictions on how content developers structure packages, either in file naming or directory structure.

2. Package Metadata (package.xml)

The metadata for a package is contained within a single XML file named “package.xml”. It must be placed in the root folder of the package and must be included for a package to be accessible by the JKO application.

package.xml must list and name each of the resources included within a package and must define a menu structure or a single resource to load automatically.

Note: Not all resources must be linked to a menu item – resources can also be linked-to within content using the Handset API.

No DTD validation is provided so attention must be paid to creating correctly-formed elements with properly named elements and attributes.

Each package definition starts with an XML encoding declaration:

The configuration of the package is then defined within a package element:

The package element is specified with two attributes:

¾ **ID** – To identify the package.

¾ **entry-point** – The ID of a resource or menu-item specified within the package to be presented to the user when the package is opened. If the a resource is specified that content will be loaded immediately. If a menu-item is specified a menu of resources can be displayed.

To provide legacy support, the application will display the first menu-item found if no ‘entry-point’ attribute has been specified in a package.

Note: Only one package element should be declared in each package.xml file.

2.1. Specifying Resources

Within package.xml content authors must use the following elements to declare, name and describe each of the resources contained within a package:

¾ **html** – An HTML document

¾ **video** – Video content

¾ **nativecode** – A link to a specialist native plugin that can be added to the application

¾ **book** – A file in an eBook format

These resources are described in this section with examples. However, each resource element must be described with the same set of six attributes:

¾ **id** - The name by which a resource is identified. A resource is referenced by concatenating the name of the element with the IDs of all parent elements, for example: "sample.faq_contentdevelopment". All IDs must be unique within a parent element (including the root).

¾ **path** – The path within the package of the resource file. Paths are relative to the root of the package.

¾ **keywords** - A space-separated list of terms to identify a resource when searching content

¾ **icon** – *Optional – currently unused but included for future extensibility*. The path to an icon image representing the resource. JPEG and PNG formats are supported – recommended size is 82 pixels x 82 pixels (other dimensions will be scaled to this). Paths are relative to the root of the package.

¾ **title** – A human-readable title for the resource.

¾ **label** – A short tag describing the resource-type.

2.1.1. HTML

To add an HTML page to a package an html element is used:

```
 path="faq/contentdevelopment/index.html”
keywords="faq content development"
icon="icons/faq/cdk.png"
title="Content Development FAQ"
label="faq" />
```

2.1.2. Video

The video resource supports media played via the mobile device’s native media player. The default format supported at present is MP4:

keywords="training development howto how to"
icon="icons/video_howto.png"
title="Development How to"
label="training" />

2.1.3. eBook

eBook content is added with the ebook resource. At present ePub and PDF formats are supported:

path="ebooks/dev_contenthandbook.epub"
keywords="development content handbook contenthandbook”
icon="icons/read_resource.png"
title="Content Development Handbook"
label="training"/>

2.1.4. Native Plugin

Where a specialist native plugin has been added to the application (and it is visible), it can be launched directly from the menu:

path="GettingStartedSearch"
keywords="development getting started”
icon="icons/read_resource.png" title="Getting Started Search"
label="training"/>

2.2. Specifying a menu of Resources

If a menu is included in the package the “menu-item” element is used to provide a menu of links to resources once the package is opened:

title="Sample">

Menu elements are specified with seven attributes:

¾ **id** – The name by which a menu item is identified. Like a resource, a menu is referenced by concatenating the name of the element with the IDs of all parent elements. Like resources all menu IDs must be unique within a parent element (including the root).

¾ **type** – The type attribute has two possible values:

– **menu** – Used when one menu item will contain other menu items to created a nested menu.

– **link** – Used to create a link to a named resource.

¾ **layout** – Used only when “type” has the value “menu”. One value is currently supported:

– **list** – Displays items within the menu as a text list with icons to the left.

¾ **link** – Used only in when “type” has the value “link”. The link should be specified as the full resource name including the package name, for example “sample.faq_contentdevelopment”.

¾ **title** – The human-readable text displayed for the menu item.

¾ **icon** – *Optional*. The path to an icon image representing the resource. JPEG and PNG formats are supported – recommended size is 82 pixels x 82 pixels (other dimensions will be scaled to this). Paths are relative to the root of the package.

¾ **enabled** – *Optional (defaults to true)*. Either “true” or “false” – allowing content authors to make some menu items inaccessible initially. These may be place-holders or content enabled programmatically based on user actions.

Linking to Resources

This is an example of a link menu to a resource within a package named “sample”:

```
 link="sample.faq_contentdevelopment" />
```

This is an example of the same link menu with an icon specified and set to inactive:

Nested Menus

This is an example nested menu – a five links to named resources (specified previously) in a menu titled “Getting Started”:

```
 type="menu"
layout="list"
title="Getting Started">

 link="sample.faq_contentdevelopment" />

 link="sample.gettingstarted_requirements" />

 link="sample.development_howto" />


 link="sample.search_gettingstarted" />
```

External References

The following example creates a link to a resource “quiz” specified in another package “framework_knowledge”:

If a content author wishes to override the title and icon specified for that external resource in its own package, replacement values can be specified in the menu item:

```
 link="quiz.framework_knowledge"
title="Test your knowledge"
icon="icons/quiz/framework_knowledge.png" />
```

2.3. Examples of package.xml

Note: in both examples comments are added using standard XML notation.

2.3.1. Directly accessing a single resource on opening

This is an example package.xml file that includes sufficient elements to load an HTML resource immediately the package is loaded. In this case the resource itself would contain menus and/or navigation tools provided by the developer:

```
 path="faq/contentdevelopment/index.html"
keywords="faq content development"
icon="icons/faq/cdk.png"
title="Content Development FAQ"
label="faq" />
```

2.3.2. Using a Menu

This is an example package.xml file that includes all the elements described in this document, using a menu. Note comments are added using standard XML notation:

```
 path="faq/contentdevelopment/index.html"
keywords="faq content development"
icon="icons/faq/cdk.png"
```

```
title="Content Development FAQ"
label="faq" />

 path="training/videos/dev_howto.avi"
keywords="training development howto how to"
icon="icons/video_howto.png"
title="Development How-To"
label="training" />

 path="ebooks/dev_contenthandbook.epub"
keywords="development content handbook contenthandbook"
icon="icons/read_resource.png"
title="Content Development Handbook"
label="training"/>


 type="menu"
layout="list"
title="Sample">

 type="menu"
layout="list"
title="Getting Started">

 link="sample.faq_contentdevelopment" />

 link="sample.development_howto" />




 link="reference.api_root" />

 link="quiz.framework_knowledge"
title="Test your knowledge" icon="icons/quiz/framework_knowledge.png" />
```

3. Package Versioning

Package version control is managed using the file “package.version” which must be included in the root of the package ZIP file.

Content Developers may populate this file with any data they wish in order to identify package versions.