



Adobe

Dolly User Manual

Technical Note #10025

Version InDesign 2.0

16 Nov 2001




ADOBE SYSTEMS INCORPORATED

Corporate Headquarters

345 Park Avenue

San Jose, CA 95110-2704

(408) 536-6000



Copyright 2001 Adobe Systems Incorporated. All rights reserved.

The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in this document. The software described in this document is furnished under license and may only be used or copied in accordance with the terms of such license.

Adobe, Adobe After Effects, Adobe InDesign, Adobe PhotoDeluxe, Adobe Premiere, Adobe Photoshop, Adobe Illustrator, Adobe Type Manager, ATM and PostScript are trademarks of Adobe Systems Incorporated that may be registered in certain jurisdictions. Macintosh and Apple are registered trademarks, and Mac OS is a trademark of Apple Computer, Inc. Microsoft, Windows, Windows 95, Windows 98, and Windows NT are registered trademarks of Microsoft Corporation. All other products or name brands are trademarks of their respective holders..

Rev #	Date	Author	Comments
0.1	16 Nov 2001	Ian Paterson.	Initial draft, based on content from the previous HTML tech-note, now deprecated.
0.2	25 Mar 2002	Ian Paterson	Updated to bring into line with latest version, and improved description of code templates a little.



Contents

Introduction.	7
Terminology and definitions	7
Key concepts.	7
Code templates	7
Getting started	8
Running on Mac OS 10	8
Running on Windows	8
Using on Mac OS 9/10.	8
Using Dolly.	8
Dolly on the Macintosh: post-processing the XML file.	9
Parameterising the generated code	9
Dolly templates	10
Dynamic tags	10
Conditional blocks	11
Summary.	11



Dolly User Manual

Introduction

This tech note describes the basics of working with Dolly the plug-in wizard. This is a Java based application that can be used on both Macintosh (OS 9 and OS 10.1) and Windows. Dolly can be extended by a third-party developer; this can be done by adding new *code templates* and/or defining new tags in a properties file that is read when a particular code template is opened. Most of the plug-ins in the InDesign 2 SDK were generated from a pair of templates, the Dialog and IfPanelMenu templates.

Terminology and definitions

- *code template*: this is the input for Dolly. A code template is code that has been parameterised with the markup tags that Dolly understands. A set of code templates are supplied with the SDK, in the folder SDK\Tools\Dolly\Templates to generate common types of plug-ins.
- *conditional replacement*: this is used to refer to substitutions made by Dolly Wizard on codeblocks containing DOLLYIF statements.
- *direct replacement*: this is when some markup in a code template such as <AUTHOR/> is replaced at code generation time by the string from the user-interface
- *dynamic tags*: these are read when Dolly is browsing a code template. The tags are defined in a file called `dynamictags.properties`.

Key concepts

Code templates

Dolly reads in a code template, makes the appropriate direct and conditional replacements, and writes to a specified output folder. The available templates for a particular technology (e.g. InDesign 2.0) will be visible in the user-interface. Dolly should ship with templates to create plugins such as a basic dialog (Dialog), menu or panel; the **IfPanelMenu** code template can create both a menu or a panel depending on parameterisation. It also has other more specialised templates, which are appropriate for creating suite implementations, and writing client code to make use of the suites (SuiteDialog, SuiteIfPanelMenu).

There are other templates for CodeSnippets (class or function), more exotic ones for adding drag-and-drop capability to a basic plug-in (AddDND), a command implementation and so on.

You will find that the IfPanelMenu templates and the Dialog templates will provide a lot of useful capability and will provide the boilerplate for many plug-ins that you write. Most of the boilerplate for the InDesign 2 SDK plug-ins was generated from one of these two code templates.

Getting started

Running on Mac OS 10

It is only necessary to double-click the file named **Dolly.jar** to run on Mac OS 10.1.x. This platform has the Java 2 run-time environment (and other Java tools) bundled as part of the operating system.

Running on Windows

You can still run Dolly on Windows by double-clicking on the **Dolly.jar** file. However, you will need to download the latest version of the Java run-time environment from <http://java.sun.com/j2se/1.4/download.html> to get the latest version of the Java run-time environment. Once you have installed this, double click on Dolly.jar.

Using on Mac OS 9/10

There is an installer for Mac OS 9. On Mac OS 9, accept options e.g. to update the Mac runtime for Java to 2.2.3 unless you have the more recent MRJ. On Mac OS 10, you already have a Java 2 platform as part of the OS bundled software. The Dolly installer will be a Classic application on Mac OS 10, but details are given below to run Dolly as a Mac OS 10 'native' Java application once it has been installed.

Using Dolly

The sequence to use Dolly is:

1. enter data, being sure to enter menu-item names that would be valid as sub-strings of a C++ name (no spaces etc),
2. validate parameters, by clicking on the button in the UI,
3. generate plug-in based off one of the code templates such as IfPanelMenu or Dialog, say
4. test plug-in, repeat steps above if necessary.

Both Macintosh and Windows plug-ins should compile cleanly and load cleanly if you have specified valid parameters. Dolly generates both code and documentation structures. The SDK

uses the documentation template created by Dolly to provide a lightweight way of documenting SDK plug-ins. You may find that it generates files into a folder such as DocSource that are not essential to your development, and you may wish to delete these to reduce the size of the generated codebase.

Dolly on the Macintosh: post-processing the XML file

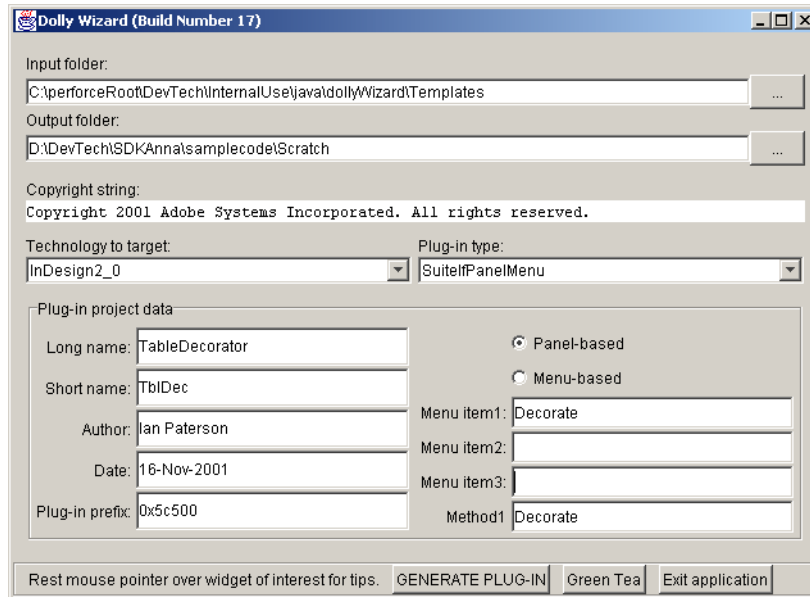
There are two steps that you should complete before you can open a CodeWarrior project on the Macintosh. The first is that you will have to set the file type of the `.mcp.xml` file that Dolly generates from the code template you have chosen. There are scripts provided with the SDK that can target Mac OS 9 or Mac OS 10, or use any tool of your choice. The file type should be 'TEXT' and the creator 'CWIE'.

Next, you have to import the `<project-name>.mcp.xml` file and save back out as an mcp project file. Be sure to quit the application by clicking on the "Close" button on the main application window and not by `Cmd-Q` to ensure that commitments you have made are correctly persisted.

Parameterising the generated code

Most of the parameters to be entered by the end-user are self-explanatory; others are explained in the section Dolly templates. Dolly will attempt to validate strings entered that could form part of C++ names; for instance, the `IfPanelMenu` template allows the user to specify names of menu items. The strings entered will form part of C++ names and the most restrictive rules for C++ variable composition apply; alphanumeric characters (or underscore) are acceptable, and variables must begin with a letter or underscore.

FIGURE 1.1 Screenshot of Dolly being used to generate a plug-in



Dolly templates

Dolly templates can contain both tags, that are the targets for replacements by the wizard, and conditional blocks, which are included if certain tags are defined. This model for code templates follows (but is not identical with) the model of code templates for Microsoft Developer Studio Custom Wizards, which have macros and conditional blocks. Dolly tags are very straightforward: for instance, instead of the author name, the template contains an expression like `<AUTHOR/>`. There is a small set of core tags, which include the following:

- `<SHNAME/>`; this specifies the short-name of the plug-in, for instance, `BscMnu` is the short-name for the `BasicMenu` plug-in.
- `<LONGPLUGINNAME/>` the long plug-in name, used for the folder name, and the name that appears in the user-interface.
- `<COPYRIGHT/>`; a user-configurable copyright string.
- `<UNIQUEPREFIXNUMBER/>`; this is the developer plug-in prefix, in the range obtained from Adobe developer support. For instance, a plug-in might have the prefix `0x5c500`. This should be entered with the `0x`, as in the screenshot in Figure 1.1.
- `<AUTHOR/>`; name of the author or company information.
- `<DATE/>`; creation date, if required. Defaults to the date today if no entry.

Dynamic tags

There are also "dynamic" tags which are read in at run-time from a properties file. These dynamic tags can appear in the user-interface with text fields to change their values, or can be

specified as mutually exclusive choices (for instance, that a template should generate a panel or a menu). That is, the user-interface elements to specify these parameters are created based on the contents of a file named "dynamictags.properties". The contents of the "dynamictags.properties" might look like this:

```
menuitem1.tag=<MENUITEM1/>
menuitem1.description=Menu Item 1:
menuitem1.default=ItemOne
```

Note that the expressions tag,description,default are not arbitrary; however, the name on the left-hand side of the expression (i.e. menuitem1 in the above expression) is arbitrary, but must be unique within the properties file. To specify that a set of choice should appear in a mutually-exclusive combination (and be represented as a group of radio-buttons), the following expression can be written:

```
# These two create a pair of grouped radio-buttons at run-time
panel.tag=<PANEL/>
panel.description=Panel based plug-in:
panel.default=false
panel.mutex=true
#
menu.tag=<MENU/>
menu.description=Menu based plug-in:
menu.default=false
menu.mutex=true
```

The most significant part of this expression is the line of the form xxx.mutex=true. If this line is present in a specification of a dynamic tag, then it will lead to a radio-button being created. There is no need to specify xxx.mutex=false in the condition where a simple text-field is appropriate.

Conditional blocks

It may be necessary to specify that some code in the template should be included when a particular condition is true, and excluded otherwise. For instance, the "IfPanelMenu" template can be used to generate a panel-based plug-in or a menu-based plug-in, dependent on end-user choice. A conditional block of code (from this template) would be bracketed like this:

```
<DOLLYIF defined="PANEL">
// code to include if this tag has been defined
</DOLLYIF>
```

The operand on the right of the attribute expression defined="xxx" refers to the tag-name once the characters such as ">", "<" and "/" have been trimmed.

Summary

Dolly takes away some of the pain from the generation of plug-ins. It is cross-platform and can be extended by third-party developers to add new code templates. It was used to generate

pretty much all of the plug-ins in the InDesign 2.0 SDK, giving them a level of code uniformity that was not present in the 1.x SDKs.