ADOBE® DEVICE CENTRAL CS4

SDK PROGRAMMER'S GUIDE

Adobe

Adobe® Device Central CS4 SDK Programmer's Guide

Adobe Systems Inc., 345 Park Avenue, San Jose, California 95110, USA.

Contents

Preface

The Adobe® Device Central Software Development Kit (SDK) allows you to create custom plug-ins that extend the tasks provided as part of Device Central's project feature. The Device Central SDK is based on ExtendScript and XML.

The SDK is available for download from http://www.adobe.com/devnet/sdks.html. It contains the elements listed in the table below. All paths are relative to the location that you choose during installation. For convenience, the SDK install location is referred to as <DC_SDK>.

<DC_SDK>/Documents/ADC SDK Guide.pdf	This programmer's guide
<DC_SDK>/Documents/ADC SDK Reference.pdf	The programmer's reference
<DC_SDK>/Documents/JavaScript Tools Guide CS4.pdf	The Adobe JavaScript reference
<DC_SDK>/Documents/adat.xsd	XML schema file
<DC_SDK>/Documents/task.xsd	XML schema file
<DC_SDK>/Sample Plugins/	Sample code that demonstrates task development
HelloWorld	A basic sample useful in getting started with the ADC SDK
CreateZip	A task plug-in that demonstrates how to use user interface components as well as the packaging of resources in an ADC task
SendToWeb	A task plug-in that sends assets to an external service.
XMP Metadata	A task plug-in that demonstrates the manipulation of SWF resources XMP metadata
<DC_SDK>/Sample Web Service/catalog	A sample Web Service application to be used with the SendToWeb sample task

In addition to the four sample task plug-ins that are part of this SDK, Adobe Device Central ships with three additional sample task plug-ins. The additional plug-ins are not packaged with the Device Central SDK, but are part of the Adobe Device Central product. You can find the code for each of these in the <REQUIRED_TASKS> folder described on page 5. This folder is where Adobe Device Central places required tasks that are part of the application. These additional plug-ins are listed in the table below.

Bluetooth	Sends a file to a Bluetooth device.
CopyFile	Copies a file to a specified file system directory.
SendToFTP	Sends a file to an FTP server.

See "Default Adobe Device Central installation folder" on page 4 for the location of the Adobe Device Central installation directory on your system.

The Adobe Device Central SDK uses the Adobe ExtendScript language and XML.

The ExtendScript language

Adobe provides an extended version of JavaScript called ExtendScript, which is used in many Adobe products including Adobe Device Central. ExtendScript is a complete implementation of ECMA JavaScript, plus additional tool and utilities. ExtendScript scripts use a `.jsx` extension, rather than the typical `.js` extension.

ExtendScript scripts can be used in either Windows or Mac OS. Your Device Central JSX scripts load automatically when you open the application by placing the scripts in the application's Tasks folder. For more information see Chapter 1, "Using the Device Central SDK."

You can use any text editor to write scripts in JavaScript. Alternatively, you can use the ExtendScript Tool Kit (ESTK). The ESTK is an interactive development environment (IDE) for JavaScript supplied with all JavaScript-enabled Adobe applications. The ESTK is the default editor for ExtendScript files that use the extension `.jsx`. See "ExtendScript installation folder" on page 5" for platform-specific information on the toolkit installation location.

For a description of the objects, tools, and utilities defined by Adobe ExtendScript, including the ExtendScript Toolkit, see the *JavaScript Tools Guide*. This guide is available in the ExtendScript application under **Help > SDK** or, alternatively, you can find the guide at `<DC_SDK>/Documents/JavaScript Tools Guide CS4.pdf`.

The XML markup language

Extensible Markup Language (XML) is a general-purpose specification for defining custom markup languages. Device Central SDK configuration files are written in XML.

The format of Device Central XML files is governed by the XML schema file `adat.xsd`.

To learn more about XML and XML schemas, go to http://www.w3.org/XML/.

About This Book

This book includes the following sections:

- Chapter 1, "Using the Device Central SDK," provides an introduction to the Device Central SDK, with the basics of how ExtendScript tasks work, and the concepts and terminology used by the Device Central SDK scripting environment.
- Chapter 2, "Getting Started: HelloWorld," takes a detailed look at a simple task that displays a greeting upon request.

  Chapter 3, "Creating a User Interface for Your Task," explains how to create and populate a dialog box in your task's `.jsx` file.

  Chapter 4, "Using ZStrings for Localization," explains how to localize your user interface for different languages.

Conventions used in this document

The following type styles are used for specific types of text.

`Monospace font`	ExtendScript code and literal values, such as function names, XML code, file names, and paths.
italic	Variables or placeholders in code. For example, in `name="`*myName*`"`, the text *myName* represents a value you are expected to supply, such as `name="Fred"`. Also indicates the first occurrence of a new term.
Blue underlined text	A hyperlink you can click to go to a related section in this book or to a URL in your web browser.
Sans-serif bold font	The names of Adobe Device Central UI elements (menus, menu items, and buttons). The **>** symbol is used as shorthand notation for navigating to menu items. For example, **Edit > Cut** refers to the **Cut** item in the **Edit** menu.

NOTE: Notes highlight important points that deserve extra attention.

Folders referenced in this document

The folders referenced in this document are summarized here. For convenience, they may be referred to by the abbreviated names shown below:

`<ADC>` is the Adobe Device Central folder/product installation directory.

`<DC_SDK>` is the Adobe Device Central SDK installation directory.

`<TASKS>` is the directory in which you should place any Adobe Device Central task plug-ins you develop.

`<REQUIRED_TASKS>` is the directory where AdobeDevice Central places required task plug-ins that are part of the application.

`<ESTK>` is the ExtendScript toolkit install location.

Default Adobe Device Central installation folder

Operating System	Path
Mac OS	`/Applications/Adobe Device Central CS4`
Windows XP	*`system drive`*`\Program Files\Adobe\Adobe Device Central CS4`
Windows Vista	*`system drive`*`\Program Files\Adobe\Adobe Device Central CS4`

User tasks folder

The operating-system-specific paths for the user tasks folder, `<TASKS>`, are given in the table below. You need to create this folder if does not already exist.

Operating System	Path
Mac OS	`/Users/`*`username`*`/Library/Application Support/Adobe/Adobe Device Central CS4/Tasks/`
Windows XP	*`system drive`*`\Documents and Settings\username\Local Settings\Application Data\Adobe\Adobe Device Central CS4\Tasks`
Windows Vista	*`system drive`*`\Users\`*`username`*`\AppData\Local\Adobe\Adobe Device Central CS4\Tasks`

Required tasks folder

The required tasks folder, `<REQUIRED_TASKS>`, can be found at the following operating-system-specific locations.

NOTE: While the plug-ins in this folder can be used for reference, the contents of this folder should not be changed.

Operating System	Path
Mac OS	`<ADC>/DeviceCentral.app/Contents/MacOS/Required/Tasks`
Windows XP	`<ADC>\Required\Tasks\`
Windows Vista	`<ADC>\Required\Tasks\`

ExtendScript installation folder

The operating-system-specific paths for the ExtendScript Toolkit installation folder, `<ESTK>`, are given in the table below.

Operating System	Path
Mac OS	`/Applications/Utilities/Adobe Utilities/ ExtendScript Toolkit CS4`
Windows XP	*`system drive`*`\Program Files\Adobe\Adobe Utilities\ExtendScript Toolkit CS4`
Windows Vista	*`system drive`*`\Program Files\Adobe\Adobe Utilities\ExtendScript Toolkit CS4`

1 Using the Device Central SDK

This chapter provides the background information you need to create your first Device Central task plug-in. It describes:

The files that make up a Device Central task plug-in. See "Task Plug-in Components" on page 6.

The order in which Device Central task plug-ins are run. See "Processing Order" on page 12.

The Device Central object model. See "Adobe Device Central Object Model" on page 9.

The `Task.jsx` interface. See "Task Plug-in Interface" on page 9.

Task Plug-in Components

This section describes how to create a Device Central task plug-in including such topics as:

Determining what components will make up your task plug-ins. See "Task plug-in files" on page 6.

Determining where to place your task plug-in's files. See "Locating your task plug-in's files" on page 11.

Organizing the files that make up your task plug-in. See "Task folder structure" on page 11.

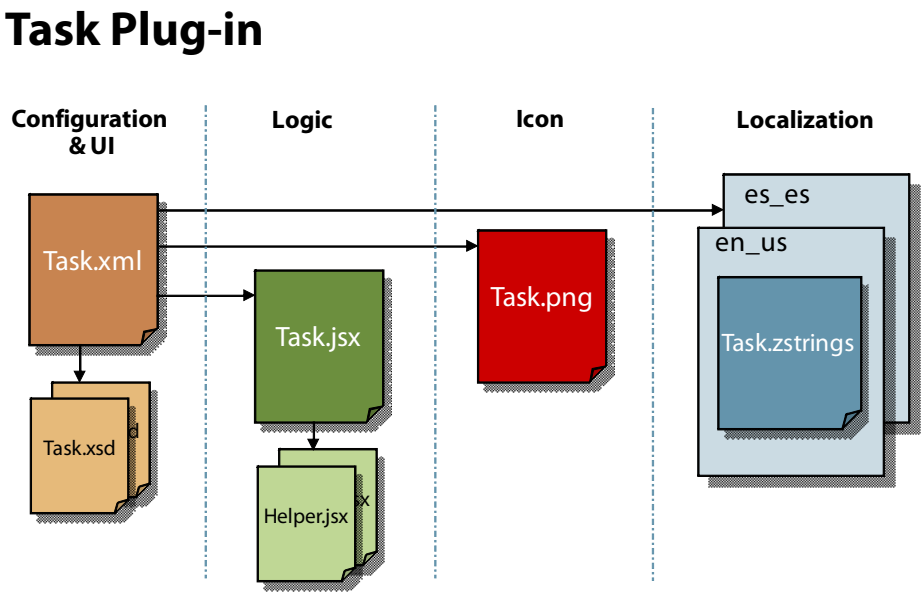
Locating your task plug-in's ZStrings. See "Localizing your task plug-in" on page 12.

Task plug-in files

Every Device Central task plug-in consists of four required components, listed below. Your plug-in also needs localization files if you intend to make it available in multiple locales. See Chapter 4, "Using ZStrings for Localization" for additional information.

Component	Description
`task.xml`	The task plug-in's configuration file
`task.png`	The task plug-in's icon to be placed inside the project toolbar
`task.jsx`	The task plug-in's script
Localization files	Folders containing ZStrings for each supported locale

A graphical view of the files that make up a task is shown below.



Task.xml

The `Task.xml` file defines general information about the task and the appearance of its user interface. The content of this file is governed by the schemas described in "Device Central Task Schema" in the *Adobe Device Central CS4 SDK Programmer's Reference*.

Broadly speaking, a Device Central plug-in XML file includes two types of information:

The `general` element contains basic information that Device Central needs, such as its title, the name of menu items used by the task, the task's icon, and the path to the task's JavaScript file.

The `dialog` element contains information about the task's user interface.

Use the `dialog` element to establish the visual representation of the configuration dialog your task needs. You can only set the visual attributes of a user interface component within the `dialog` element. The Device Central API lacks the ability to dynamically add UI components during runtime. Therefore, you must define all components your task needs here. See Chapter 3, "Creating a User Interface for Your Task," for more information.

Task.png

The `Task.png` file is the graphical representation of the task in the project window. The icon must be a PNG or BMP file, 16 by 16 bits in size.

Task.jsx file

The `Task.jsx` file contains the logic required for the plug-in and handles any user input. Broadly speaking, this file consists of three mandatory functions along with any other helper functions those functions call. Each of these functions has access to the task and project objects. For more detailed information see "Task Plug-in Interface" on page 9.

Task plug-in libraries

As previously stated, plug-ins use ExtendScript, an extended implementation of the JavaScript language used by many Adobe applications that provide a scripting interface. Therefore, when developing a new plug-in, you can make use of the features provided by the JavaScript language as well as the additional features and utilities provided by ExtendScript. Adobe Device Central also provides a library of objects to assist in the creation of plug-ins.

The following sections describe how to work with each of these libraries:

JavaScript

ExtendScript

JavaScript

When creating a plug-in, you can use the familiar JavaScript constructs including variables, functions, statements, operators, control structures, and exception handling. JavaScript also provides a series of core classes, including `String`, `Array`, `Boolean`, and `Number`, that are available for use in your plug-ins.

For detailed documentation on the JavaScript language or information on how to use it, see publicly available web resources or any of numerous works on this subject, including the following:

The public JavaScript standards organization website: www.ecma-international.org

JavaScript: The Definitive Guide, 5th Edition; Flanagan, D.; O'Reilly 2006; ISBN 0-596-10199-6

JavaScript Programmer's Reference; Horn, S; Wrox; ISBN 0-470344-72-5

JavaScript Bible. 6th Edition; Goodman, D., Eich, B., and Morrison, M.; John Wiley and Sons 2007; ISBN 0-470-06916-3

ExtendScript

These are some of the features and utilities provided by Adobe's extended implementation of JavaScript that can be used when creating a Device Central task plug-in.

Cross-platform file-system access through the `File` and `Folder` classes.

Communication between Adobe message-enabled applications through the defined `BridgeTalk` class (that is, Adobe Flash, Adobe Photoshop®, and Adobe Illustrator®).

The `ExternalObject` class provides a way to extend Device Central JavaScript DOM with your own C or C++ shared libraries.

The `XML` object allows you to process XML within your script.

Scripting support for the manipulation of XMP metadata.

Localization of text.

For more detailed information and sample code for the features and utilities provided by ExtendScript see the *JavaScript Tools Guide*.

Adobe Device Central Object Model

The Device Central Object Model makes available a number of objects that facilitate the development of plug-ins. For complete information on the Device Central objects, see "Adobe Device Central DOM Object Reference" in the *Adobe Device Central CS4 SDK Programmer's Reference*.

Device Central objects

A `Project` object provides information about a Device Central project.

A `Task` object contains information regarding a Device Central task plug-in.

A `Device` object provides device profile information.

User interface objects

A `Control` object represents a distinct user interface widget.

A `StatusDialog` object represents a user interface dialog box that informs the user of task execution progress.

Utility

A `ZIPFile` object allows the creation and management of archive files.

A `ListItem` object allows the creation of objects with dynamic properties.

A `URLStream` object provides a mechanism to communicate with external resources over the network.

A `Bluetooth` object is used to manage and communicate with Bluetooth-enabled devices.

For additional information of each of the objects described above, see "Adobe Device Central DOM Object Reference" in the *Adobe Device Central CS4 SDK Programmer's Reference*.

Task Plug-in Interface

The `Task.jsx` script contains all the processing required for the task plug-in, including the handling of user input.

Broadly speaking, the `Task.jsx` file consists of three mandatory functions along with any other helper methods those functions call. Each of these functions has access to the task and project objects.

Mandatory functions

The mandatory functions are listed in the table below.

Function	When it's called?	Use
`configure(toolbar)`	Called before the **Task** configuration dialog is presented to the user. For tasks initiated from the toolbar, called each time the task is selected to run. The argument `toolbar` is set to `true` in this case. For tasks initiated from the **Task** section of the project, called: ➢ Every time the user selects the **Run task** option, if the **Show dialog before running task** option was checked at the time of task creation. Every time the user selects the **Edit task** option. The argument `toolbar` is set to `false` in these cases.	Sets up the user interface elements in the configuration dialog before it is displayed.
`execute()`	Called when the user requests that the task run. The user can click one of the tasks in the toolbar, and then click the configuration dialog's **Run** button, or can click the **Run task** icon in the **Tasks** section after a saved task is selected.	Retrieves user interface input from the configuration dialog and processes any request.
`install()`	Called each time an instance of a task plug-in is created. For tasks in the project toolbar, this happens once when the project is first created or opened. For tasks in the **Tasks** section of the project, `install()` is called: When the user creates a new instance of the task by choosing that task from the **New Task** menu. Each time a project with that saved task is re-opened.	Sets up default task information.

Global variables

All functions in the `Task.jsx` file have access to the global variables `task` and `project`.

Variable	Type	Description
`task`	`Task`	An instance of the plug-in currently in use that provides access to its components and allows you to interact with the client machine. See "Adobe Device Central DOM Object Reference" in the *Adobe Device Central CS4 SDK Programmer's Reference* for more details.
`project`	`Project`	An instance of the current active project that provides access to its resources and devices. See "Adobe Device Central DOM Object Reference" in the *Adobe Device Central CS4 SDK Programmer's Reference* for more details.

Helper methods

In addition to the three required methods (`install()`, `configure()`, and `execute()`), developers can add any number of helper methods and variables to a task plug-in's main script to ease the development of the plug-in. Developers can also use a modular approach to the development of plug-ins by using ExtendScript directives to include external scripts. For example:

```
#include "myfile.jsx"
```

See "Modular Programming Support" section in the *JavaScript Tools Guide* for more details.

Locating your task plug-in's files

Device Central looks for plug-ins in the two folders listed below.

The `<REQUIRED_TASKS>` folder and subfolders is the first hierarchy level. This folder is reserved for use by Adobe for task plug-ins that ship with Device Central.

The Adobe Device Central CS4 user data folder `<TASKS>` and subfolders in the first hierarchy level. This is where custom user-created or third-party task plug-ins should be placed.

For more information of how Device Central manages plug-ins, see "Processing Order" on page 12."

Task folder structure

All of the core files that make up a task plug-in reside in a single folder. Each of a task plug-in's component files must appear within that folder along with a folder that contains localization files. If a complex plug-in requires subfolders to organize its files, all paths within the files in the subfolder should be relative to the top level `task.xml` file.

Typically, the files `Task.xml`, `Task.png`, and `Task.jsx` are placed at the first level within the task folder. In addition, the first level within that folder contains one folder for each locale supported by the plug-in. Each of those folders contains a ZString dictionary file (`Task.zstring`) with the localized text.

The name of the ZString folder for each locale is the concatenation of an abbreviation for the language, followed by an underscore, followed by an abbreviation for the country. For example, the folder `en_us` would hold ZStrings in English for use in the United States. Similarly, the folder `ja_jp` would hold ZStrings in Japanese for use in Japan.

The organization of a possible `Tasks` folder is illustrated below.

```
Tasks-->CopyFile-->copy.png
      |            |
      |            FileCopy.jsx
      |            |
      |            FileCopy.xml
      |
      |
      -->HelloWord-->HelloWorld.jsx
                     |
                     HelloWorld.png
                     |
                     HelloWorld.xml
                     |
                     en_GB-->HelloWorld.zstrings
                     |
                     en_US-->HelloWorld.zstrings
                     |
                     en_ES-->HelloWorld.zstrings
```

Localizing your task plug-in

The Device Central task plug-in architecture supports the ZString format.

Each task plug-in must supply its own localization files.

Any ZStrings that appear inside the XML are localized automatically if a dictionary for the locale required exists.

Strings appearing inside a JavaScript file must be localized using the global `localize()` function found in the ExtendScript functionality. For more information on how to use ZString to localize your plug-in see Chapter 4, "Using ZStrings for Localization."

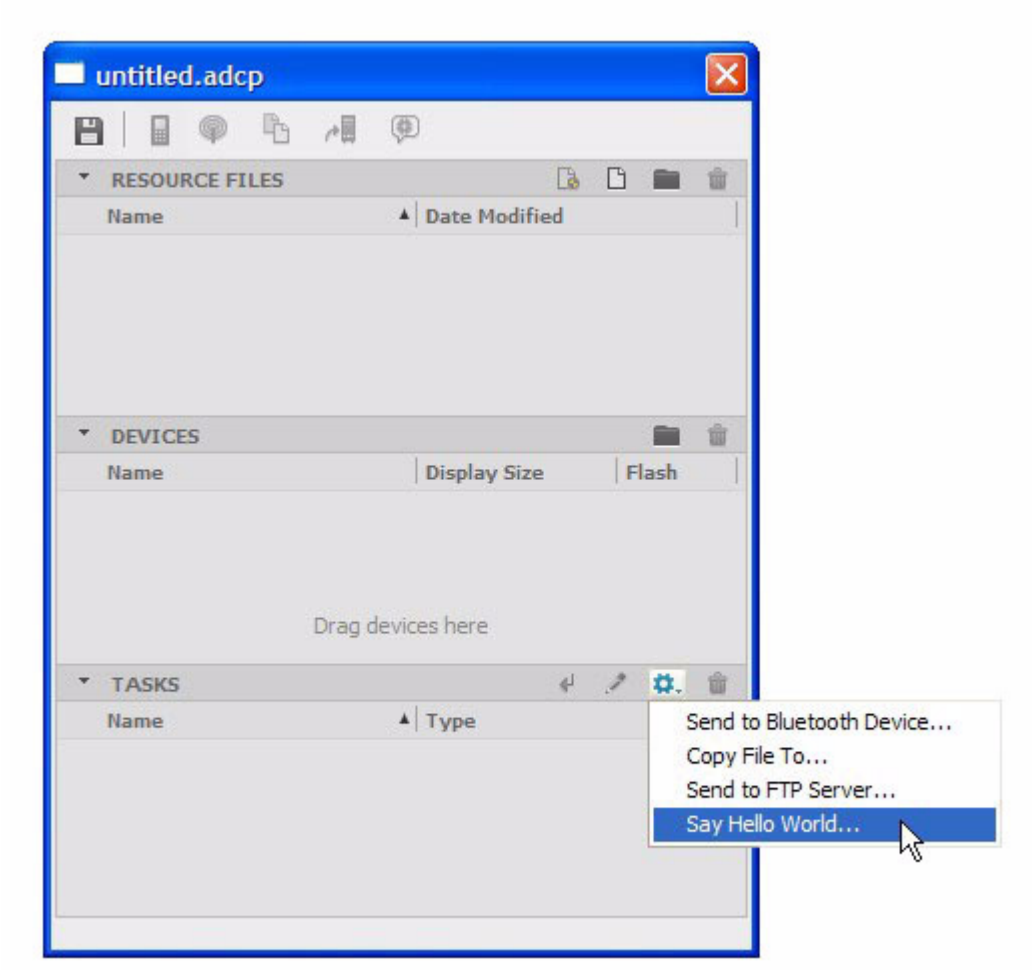
Processing Order

When starting up tasks, Device Central first looks for plug-ins in the `<REQUIRED_TASKS>` folder. Only after all those tasks are loaded does it look for task plug-ins in the user data folder. This ensures that the common tasks maintained by the Device Central team—`Bluetooth`, `CopyFile`, and `SendToFTP`—are loaded before any user task plug-ins.

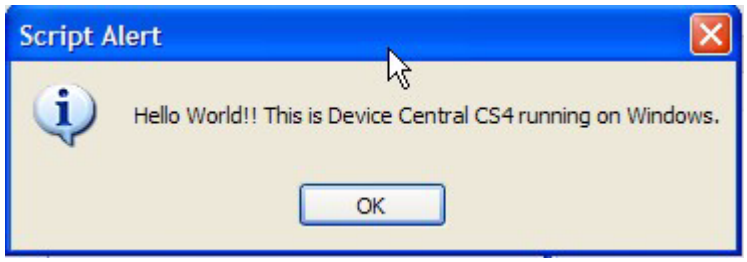
2 Getting Started: HelloWorld

This chapter helps you get started in creating Device Central task plug-ins by walking you through the creation of the `HelloWorld` task plug-in. This task is added to the list of existing tasks as soon as it is installed in the user's custom tasks folder.

You can find all the code for the `HelloWorld` task plug-in in the Device Central SDK sample plug-ins directory (`<DC_SDK>/SamplePlugins`).



When the user chooses this task from the task list, a greeting is displayed. Sample output from this task is shown below.



The `HelloWorld` task plug-in works in either a Windows or a Macintosh OS environment. You can use the IDE of your choice, but the use of the ExtendScript Toolkit is recommended.

While the `HelloWorld` task plug-in is simple, it illustrates key techniques you need to master in order to create your own, more sophisticated tasks. In particular, this example shows you how to:

Determine what files are needed for the `HelloWorld` task plug-in. See "Plug-in Files" on page 14.

Install the `HelloWorld` task plug-in. See "Installing the Plug-in" on page 15.

Configure the `HelloWorld` task plug-in. See "Configuring a Task" on page 15.

Create an icon for your task plug-in. See "Creating an Icon" on page 17.

Create an ExtendScript script for the `HelloWorld` task plug-in. See "Creating a Script" on page 18.

Localize the `HelloWorld` task plug-in. See "Localizing Your Task" on page 19.

Run and debug your task plug-in. See "Testing Your Task" on page 20.

All of the concepts and techniques described in this chapter are discussed in great detail in elsewhere in this book.

Plug-in Files

Building your `HelloWorld` task plug-in requires that you create several files:

`HelloWorld.xml` contains the task plug-in's description information. This is the information Device Central uses to locate and set up your task for user viewing. Among other things, it references the task's icon, script, and localization files described below.

See "Configuring a Task" on page 15 to learn how to build your plug-in's description file.

`HelloWorld.png` is the task's icon. This is the image that the users see on the toolbar.

See "Creating an Icon" on page 17 to learn how to create the `HelloWorld` icon.

`HelloWorld.jsx` is the ExtendScript file for this plug-in. The logic necessary to carry out the task's work can be found in this file.

See "Creating a Script" on page 18 to learn how to write the JavaScript code for the `HelloWorld` task plug-in.

In addition, because the `HelloWorld` task plug-in is localized for three countries—the United States, Great Britain, and Spain—you need to create a set of ZStrings for each of these locales.

Installing the Plug-in

In order for Device Central to discover your plug-in files, they should be placed in a special directory. Installing your task means creating the appropriate folder and placing the required files in that folder.

1. Begin by locating the task directory where you will place your own task's folder. Follow the directions below for your operating system:

   On Windows Vista, place the task's folder in `Users\username\AppData\Local\Adobe\Adobe Device Central CS4\Tasks`.

   On other Windows platforms, place the task's folder in `C:\Documents and Settings\Users\username\LocalSettings\Application Data\Adobe\Adobe Device Central CS4\Tasks`.

   On the Mac OS, place the task's folder in `/Users/username/Library/Application Support/Adobe Device Central CS4/Tasks`.

   For simplicity, these directories are referred to as `<TASKS>` in the remainder of this book.

2. Create a new folder named `HelloWorld` for your task's files.

Having installed your task, you are now ready to create the `HelloWorld` task's description file.

Configuring a Task

The description file, `HelloWorld.xml`, provides Device Central with information it needs to correctly run your task.

1. Use any text editor to create the `HelloWorld.xml` file.
2. Save `HelloWorld.xml` in `<TASKS>/HelloWorld`.
3. Add the XML version number and the encoding on the first line of `HelloWorld.xml`.

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

4. Add the `task` element in the `adat` namespace along with its attributes.

```
<adat:task xmlns:adat="http://ns.adobe.com/devicecentral/task/"
          guid="5cff2c68-96b3-11dc-8314-0800200c9a7e"
          version="1">


</adat:task>
```

   `adat` is the default prefix for the `task` namespace.

   The `xmlns:adat` element defines the XML namespace used by the task. The value is `http://ns.adobe.com/devicecentral/task/`.

   The `task` element has two attributes:

   The `guid` attribute has the value `5cff2c68-96b3-11dc-8314-0800200c9a7e`.

   This value is a random 128-bit Global Unique Identifier (GUID) that uniquely identifies the task. GUIDs consist of a string of random hexadecimal digits that are displayed in successive groups of

8, 4, 4, and 12 hexadecimal digits with groups separated by the hyphen (-) character. The pattern is thus `xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx`, where each `x` is a hexadecimal digit.

Any value that uniquely identifies your plug-in is valid here even if it does not conform to the GUID standard.

The `version` attribute has the value `1`, signifying that this is the first version of the `HelloWorld` task.

5. When the `task` element is defined, add the first of its two children, `general`. This element needs a number of child elements whose content defines basic information Device Central uses to configure this task.

   `HelloWorld.xml` now has the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<adat:task xmlns:adat="http://ns.adobe.com/devicecentral/task/"
          guid="5cff2c68-96b3-11dc-8314-0800200c9a7e" version="1">
    <general>

    </general>
</adat:task>
```

6. Add each of the children of the `general` element.

   The elements to add are:

   `title`, whose `value` attribute is the text string that appears in the project window when you create a new task. The type of a task is the title element.

   `menuItem`, whose `value` attribute is the localized text string that appears on the Device Central's **Project > New Task** menu.

   `toolTip`, whose `value` attribute is the localized text string to be used for the task's tooltip.

   `script`, specifying the task's script path, relative to the XML file.

   `icon`, the path of the icon file, relative to the XML file.

   `localization path`, the path relative to the XML of the task's localization files.

   `toolBar`, set to `true` if the task should appear in the Device Central project toolbar. Otherwise it is `false`.

   The `general` element now looks like this:

```
<general>
       <title value="$$$/HelloWorld/Title=Hello World"/>
       <menuItem  value="$$$/HelloWorld/Menu=Say Hello World..."/>
       <toolTip value="$$$/HelloWorld/ToolTip=Say Hello World"/>
       <script path="HelloWorld.jsx"/>
       <icon path="HelloWorld.png"/>
       <localization path="HelloWorld.zstrings" />
       <toolbar value="true" />
</general>
```

7. Add the `dialog` element, the second of the task element's two children. This element sets up the initial user interface for the `HelloWorld` task.

This element has attributes used to determine height, width, and minimum height and width of the configuration dialog box.

```
<dialog height="250" width="600" minHeight="250" minWidth="600">
</dialog>
```

8. Add the `dialog` element's child element `container` along with its attributes `layout` and `sizex`. These attributes determine the layout and the distribution of the user interface elements in the dialog.

```
<dialog height="250" width="600" minHeight="250" minWidth="600">
    <container layout="vertical" sizex="scale">
    </container>
</dialog>
```

9. Add the container element's four child elements. Three are `staticText` elements with differing attribute and attribute values. One is a `textField` element that allows a place for the user to enter a personalized message.

   The `staticText` elements provide display information that can be set either in the XML file itself or dynamically in the code.

```
<dialog height="250" width="600" minHeight="250" minWidth="600">
       <container layout="vertical" sizex="scale">

       <staticText width="300" alignment="left">
              <label value="$$$/HelloWorld/UI/EnterName=
                 Enter your name for a personalised message:" />
       </staticText>

       <textField id="name" width="250" sizex="scale"/>

       <staticText id="installmessage" width="430" alignment="left">
          <label value="" />
       </staticText>

       <staticText id="confmessage" width="430" alignment="left">
          <label value="" />
       </staticText>

       </container>
   </dialog>
```

10. The `HelloWorld.xml` description file is complete. Save and close the file.

Creating an Icon

Each task plug-in that appears in the task bar needs an icon to represent that task to the user. An icon is a 16x16 bitmap image in the Portable Network Graphics (PNG) format.

You can create the `HelloWorld` icon using any bitmap editor.

1. Create a file that is 16 pixels wide and 16 pixels high.
2. Create an icon that represents your `HelloWorld` task.

3. Save the icon  under the name `HelloWord.png`.

Creating a Script

The actual `HelloWorld` script is the `HelloWorld.jsx` file. This file contains three required functions:

The `install()` function is called each time a new instance of a task is created.

The `configure()` function does any work necessary to configure the task. It is called before the **Configuration preferences** dialog is displayed at the time that a task is run.

Here `HelloWorld` displays a message in configuration method's dialog with the time of the last configuration.

The `execute()` function does the actual work of the task. It is here that the `HelloWorld` greeting is coded.

See "Task.jsx file" on page 7" for additional information on these functions.

Create the ExtendScript file

1. Using the ESTK or any other any text editor, create the file `HelloWorld.jsx`. Save the file in `<TASKS>/HelloWorld/HelloWorld.jsx`.

2. Add the code for the required `install()` function. The function displays a message in the configuration dialog indicating the time the task plug-in was installed.

```
function install()
{
    /* Set the a message in the configuration dialog with the time the
       was installed */

    var message = localize("$$$/HelloWorld/Installed=
       This task was installed on %1." , now());
    task.getStaticText("installmessage").label = message;
}
```

3. Add the code for the required `configure()` function.

```
function configure(toolbar)
{
    /* Set the message in the configuration dialog with the time of
    last configuration of the task */
    var message = "";

    if (toolbar)
        message ="$$$/HelloWorld/Configured=
            This task was configured on %1 from the toolbar.";
    else
        message = "$$$/HelloWorld/Configured=This task was configured on %1.";

    task.getStaticText("confmessage").label = localize(message, now());
}
```

4. Add the code for the `execute` function. This is where the actual work of the task plug-in is done.

```
function execute()
```

```
{
    /* Read the user's input */
    var name = task.getTextField("name").text;

    /* When the name is not available greet the world */
    if (name == undefined || name.length == 0)
        name = localize("$$$/HelloWorld/World=World");

    var os = localize(task.isOSMacintosh ? "$$$/HelloWorld/OSMac=
           Mac OS" : "$$$/HelloWorld/OSWindows=Windows");


    alert(localize("$$$/HelloWorld/Hello=
        Hello %1!! This is Device Central CS4 running on %2.", name, os));

}
```

5. Create the `now()` method, which is used to get the date displayed by `install()` and `configure()`.

```
function now()

{

    return new Date().toString();

}
```

Coding of the `HelloWorld` script is now complete.

Localizing Your Task

The `HelloWorld` task uses text strings when constructing the information that appears in its dialog boxes. By localizing the task plug-in, you make it possible for users in multiple locales with different languages to easily use the task. For more information, see Chapter 4, "Using ZStrings for Localization."

The work of localization happens in three places:

- The `HelloWorld.xml` description file references the task plug-in's ZStrings.
- The `HelloWorld.jsx` script references the task plug-in's ZStrings.
- The task plug-in's directory contains three subdirectories, one for each of the supported locales.

Within each of the directories, you need to save a `HelloWorld.zstring` dictionary file with the text string keys and localized values for each language.

For example, the table below lists three countries, their locale names, and a sample ZString that might appear in the `HelloWorld.zstring` dictionary file.

Country	Locale String	Sample ZString
Great Britain	`en_GB`	`"$$$/HelloWorld/UI/EnterName=Enter your name for a personalised message:"`
United States	`en_US`	`"$$$/HelloWorld/UI/EnterName=Enter your name for a personalized message:"`
Spain	`es_ES`	`"$$$/HelloWorld/UI/EnterName=Escribe tu nombre para recibir un mensaje personalizado:"`

NOTE: See "Locales" in the *Adobe Device Central CS4 SDK Programmer's Reference* for the full list of Adobe supported locales and their associated codes.

Testing Your Task

To test your `HelloWorld` task plug-in:

1. Start up Device Central.
2. Select **New project** from the **File** menu.
3. Choose the `Say Hello World` task from the project's **New Task** menu and save the task.
4. Select **Run Task** and verify that the `Hello World` greeting appears in an alert box.

Troubleshooting

If the `HelloWorld` task plug-in is properly coded and installed, the following should all be true:

Your task appears in the menu bar.

If not, your task plug-in might not have been loaded.

The `HelloWorld` icon appears on the task bar.

If the icon fails to appear, see "Debugging icon problems" on page 20.

The `install()` message appears when you create a new `HelloWorld` task.

The `configure()` function's message appears in the configuration dialog when the saved task is run unless you have turned off the **Show this dialog before running** option when creating the task.

If the message fails to appear, see "Debugging configuration problems" on page 20.

The `Hello World` greeting appears in the configuration dialog when you run the `HelloWorld` task.

If the greeting fails to appear, see "Debugging functionality" on page 21.

Debugging configuration problems

If the configure alert message fails to appear:

1. Verify that you have placed the appropriate code in the `HelloWorld.jsx` file.
2. Verify that `HelloWorld.xml` exists in the correct location and has appropriate content.

Debugging icon problems

1. Verify that your icon file exists and is in the appropriate place in the `HelloWorld` task plug-in directory.
2. Verify that the icon file has the correct name and that it matches that given in `HelloWorld.xml`.
3. Verify that the icon file is in the PNG format.

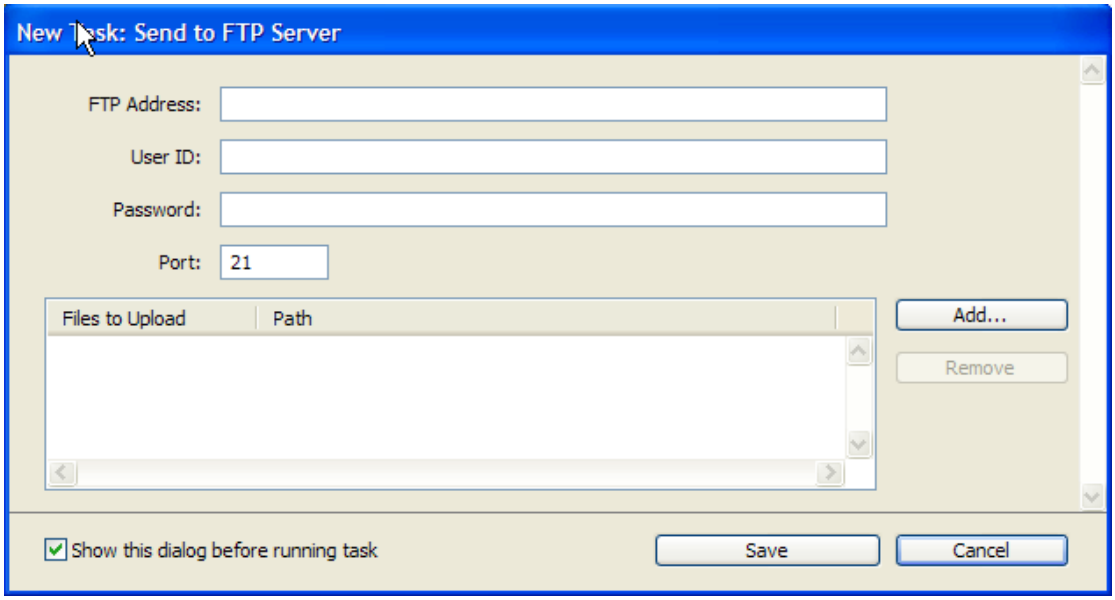
Debugging functionality

1. Verify that you have placed the appropriate code in the `HelloWorld.jsx` file.

3 Creating a User Interface for Your Task

An Adobe Device Central task's user interface consists of a customizable dialog box that can display information to the user and retrieve user input prior to that task's execution.

For example, the custom dialog box for the `Send to FTP Server` task plug-in provided with Adobe Device Central is shown below. This dialog box contains a series or components or widgets that allow a user to input connection details and to indicate the files that need to be uploaded by the task plug-in. You can create your own custom dialog boxes using the information provided below.



The Device Central Control object manages and customizes the available user-interface widgets in a task's configuration dialog.

There are two places you can manage a task plug-in's user interface:

From the task plug-in's XML file:

You can define the components that make up a task plug-in as well as the look and feel of that task within the dialog element.

From the task plug-in's JSX file:

You can dynamically modify some of the attributes of the user-interface controls defined in the task plug-in's XML file within the task plug-in's JSX file.

Creating User Interface Objects

All user interface components must be defined in the `Task.xml` file. User interface widgets cannot be instantiated dynamically from the task plug-in's JSX file.

The following XML code defines a basic user interface for a task plug-in. It specifies the dimension of a dialog box, and defines a container and two controls. The controls consist of static text and an input text field that are laid out vertically within the container.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<adat:task xmlns:adat=”http://ns.adobe.com/devicecentral/task/“
guid=”5cff2c68-96b3-11dc-8314-0800200c9a7e” version=”1”>
<general>
    <!—task configuration elements not shown -->
</general>
<dialog height="250" width="600" minHeight="250" minWidth="600">
    <container layout="vertical" sizex="scale">
        <staticText width="300" alignment="left">
        <label value="$$$/HelloWorld/UI/EnterName=
            Enter your name for a personalised message:" />
        </staticText>
        <textField id="name" width="250" sizex="scale"/>
    </container>
</dialog>
</adat:task>
```

Defining user interface elements

The elements that can be used to define a task plug-in's user interface are described in the `task.xsd` schema that is part of the Adobe Device Central SDK.

There are three categories of user interface elements described in the sections that follow:

Dialog elements

Control elements

Other elements

Dialog elements

A `dialog` element represents the configuration dialog window and contains a sequence of `control` elements. It has the attributes listed in the table below.

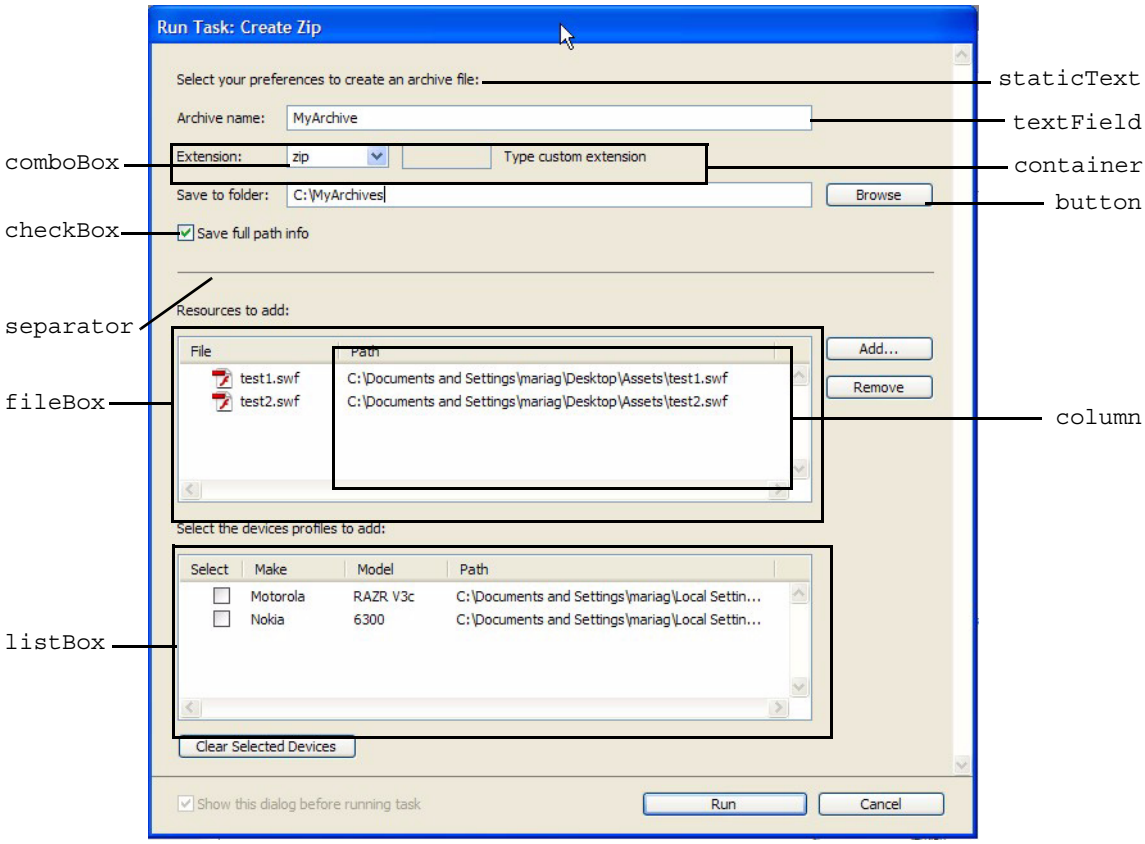
Attributes	Data type
`height`	`xs:integer`
`width`	`xs:integer`
`minHeight`	`xs:integer`
`minWidth`	`xs:integer`

Control elements

The `control` elements all represent mechanisms via which dialog box widgets are initially defined. `control` elements are accessible from the script file as an instance of the Adobe Device Central JavaScript `Control` object. The varieties of control elements available for use in Adobe Device Central dialog boxes are listed in the table below.

Element	Note
button	Contains one label element.
checkBox	Contains one label element.
comboBox	-
container	Contains a sequence of control elements.
fileList	Contains one columns element.
listBox	Contains one columns element.
passwordField	Contains one text element.
separator	-
staticText	Contains one label element.
textField	Contains one text element.

The user interface appearance of a number of the elements listed in the table above are shown in the figure below. See "Other elements" on page 25 for information on the elements referenced in the Notes above.



Control element attributes

The `control` element attributes are listed below along with their data type. All but four of the attributes apply to all of the control elements. Four of the attributes are specific to the elements named.

NOTE: An `NCName` is a string that does not contain colons.
See http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#NCName for more details.

Attribute	Data type	Applies to
`alignment`	`xs:NCName`	`staticText`
`checked`	`xs:boolean`	`checkBox`
`enabled`	`xs:boolean`	All control objects
`height`	`integer`	All control objects
`icon`	`xs:NCName`	`fileList`
`id`	`xs:NCName`	All control objects
`layout`	`vertical|horizontal`	`container`
`onClick`	`xs:NCName`	`button`, `checkbox`, `comboBox`
`sizex`	`left|scale|right`	All control objects
`width`	`xs:integer`	All control objects

Other elements

The remaining elements and their attributes are listed in the table below.

Element	Note
`columns`	Contains a sequence of `column` elements.
`column`	-
`label`	-
`text`	-

Attribute	Data type	Applies to
`width`	`xs:integer`	`column`
`columnText`	`xs:string`	`column`
`dataField`	`xs:NCName`	`column`
`value`	`xs:NCName`	`label`, `text`

Here is an example showing the use of these elements within a `listBox`.

```
<listBox id="devices" width="550" height="140" sizex="scale" selectable="true">
    <columns>
       <column dataField="selected"
              columnText="$$$/CreateZip/UI/Select=Select" width="50" />
        <column dataField="make"
              columnText="$$$/CreateZip/UI/DeviceMake=Make" width="80" />
         <column dataField="model"
              columnText="$$$/CreateZip/UI/DeviceModel=Model" width="80" />
        <column dataField="path"
              columnText="$$$/CreateZip/UI/DevicePath=Path" width="300" />
    </columns>
</listBox>
```

Managing User Interface Objects

While you cannot create a new user interface object in the `Task.jsx` script, you can access some object properties and make modifications to the content of existing controls from within that script. The example that follows illustrates some of the possible changes.

```
//Return a control object representing the text field with id "name"
var myTextField = task.getTextField("name");

//Get the input entered by the user in the text field
var myName = myTextField.text;

//Set the text field text to a new value
myTextField.text = "My new name";

//Disable the text field in the dialog
myTextField.enabled = false;
```

See "Adobe Device Central DOM Object Reference" the *Adobe Device Central CS4 SDK Programmer's Reference* for more information on working with the Control object model from within the task plug-in's script.

For an example of how to use the available `control` elements, see the `CreateZip` sample code that is included with the Adobe Device Central SDK. That code makes use of most the different control objects in the dialog box configuration.

4 Using ZStrings for Localization

Device Central task plug-ins support the ZString format. *ZStrings* are an Adobe convention for defining localization strings.

To use ZStrings in Adobe Device Central, you identify a string according to its usage in the user interface, and specify it in the The ZString Format. This enables Device Central to look up language-specific versions of the string to display to the user.

In Device Central, you place ZStrings in a language-specific folder. That folder is a subfolder of the `Task` folder. Each folder's name is a locale name representing a country-language combination. See "Locale Codes" in the *Adobe Device Central CS4 SDK Programmer's Reference* for more details.

Resolution of ZStrings depends on dictionary files that you supply, which contain the mappings from the ZString path to the localized string.

The ZString Format

The format of a ZString is:

```
$$$/ZString_path/stringKey=defaultValue
```

A description of the contents of a ZString follows.

`$$$`	The ZString marker is always required to identify a ZString and distinguish it from any other 8-bit ASCII string.
/ZString_path/ stringKey=	The path and key uniquely identify a specific string, and are used to look up the translation in a dictionary file that you provide with your task plug-in. The path is a series of 7-bit ASCII character strings separated by the slash (/) character. You can use any strings you wish with the exception of white space. The last element of the path is a specific key name, which is separated from the default value by an equal sign (=). The path groups a set of properties. For example, you might use a unique path for a particular task plug-in, and within that task plug-in further group all strings that appear in a particular dialog. Each task plug-in has its own mapping of the context paths, so your paths do not conflict with those used by other task plug-ins, or by Device Central itself.
defaultValue	The string following the separator (=) is the default display string to use for this ZString. If no matching key exists in the active localization dictionary (or if no appropriate dictionary is found), this value is displayed to the user. Strings values used in ZStrings can contain escape sequences to indicate certain characters; see "Working with ZStrings" on page 28.

ZStrings can be enclosed in single or double quotes. For example:

```
"$$$/MyTask/UI/sectionName=Description"
'$$$/MyTask/UI/Title=Document Title:'
```

Working with ZStrings

The Adobe Device Central task plug-in architecture supports the ZString format. Note that:

Each task plug-in provides its own localization files.

ZStrings inside the XML are localized automatically.

To localize strings inside a JavaScript file, you must call the global `localize()` function ExtendScript functionality.

Using ZStrings in Task.zstrings

To create a localized dictionary, you must create a directory whose name matches the locale name for each country and language combination for which you are localizing. See "Locale Codes" in the *Adobe Device Central CS4 SDK Programmer's Reference* for a list of these codes.

For example, a task plug-in that is localized for U.S. English, British English, and Spanish (Spain) would need the following three directories:

```
en_GB

en_US

es_ES
```

Within each folder, you need a `Task.zstrings` file containing a list of needed ZStrings. A sample ZString file is shown below.

```
"$$$/HelloWorld/Title=Hello World"
"$$$/HelloWorld/Menu=Say Hello World..."
"$$$/HelloWorld/ToolTip=Say Hello World"
"$$$/HelloWorld/UI/NotInstalled=Installation has not been called yet."
"$$$/HelloWorld/UI/NotConfigured=Configuration has not been called yet."
"$$$/HelloWorld/UI/EnterName=Enter your name for a personalised message:"
"$$$/HelloWorld/Installed=This task was installed on %1."
"$$$/HelloWorld/Configured=This task was configured on %1."
"$$$/HelloWorld/ConfiguredToolbar=This task was configured on %1 from the toolbar."
"$$$/HelloWorld/World=World"
"$$$/HelloWorld/OSMac=Mac OS"
"$$$/HelloWorld/OSWindows=Windows"
"$$$/HelloWorld/Hello=Hello %1!! This is Device Central CS4 running on %2."
```

Using ZStrings in Task.xml

The following is a sample `Task.xml` file that uses ZStrings to facilitate localization.

For example, the following attribute value assignment uses an ordinary string.

```
<title value=”Hello World”/>
```

This attribute value assignment uses a ZString to facilitate localization.

```
<title value="$$$/HelloWorld/Title=Hello World"/>
```

The full example follows.

```
<adat:task xmlns:adat="http://ns.adobe.com/devicecentral/task/"
    guid="5cff2c68-96b3-11dc-8314-0800200c9a7e" version="1">
    <general>
        <title value="$$$/HelloWorld/Title=Hello World"/>
        <menuItem  value="$$$/HelloWorld/Menu=Say Hello World..."/>
        <toolTip value="$$$/HelloWorld/ToolTip=Say Hello World"/>
        <script path="HelloWorld.jsx"/>
        <icon path="HelloWorld.png"/>
        <localization path="HelloWorld.zstrings" />
        <toolbar value="true" />
    </general>
    <dialog height="250" width="600" minHeight="250" minWidth="600">
        <container layout="vertical" sizex="scale">
           <staticText width="300" alignment="left">
            <label value="$$$/HelloWord/UI/EnterName=
                   Enter your name for a personalised message:" />
           </staticText>
           <textField id="name" width="250" sizex="scale"/>
           <staticText id="installmessage" width="430" alignment="left">
            <label value="" />
            </staticText>
            <staticText id="confmessage" width="430" alignment="left">
            <label value="" />
            </staticText>
        </container>
    </dialog>
</adat:task>
```

Using ZStrings in Task.jsx

The following code fragment from a `Task.jsx` file illustrates the use of the `localize()` function.

```
task.getStaticText("message").label =
       localize("$$$/HelloWorld/Message=Hello World");
```

Index

O

P

R

S

T

U

W

X

Z