ADOBE® DEVICE CENTRAL CS4

SDK PROGRAMMER'S REFERENCE

Adobe

Adobe® Device Central CS4 SDK Programmer's Reference

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

Contents

Preface **4**

About This Book 4

Document Conventions 5

Where to Go for More Information 7

1 Adobe Device Central DOM Object Reference **8**

Bluetooth 8

Control 13

Device 15

ListItem 15

Project 16

StatusDialog 19

Task 20

URLStream 23

ZIPFile 27

2 Device Central Task Schema **29**

adat.xsd 29

task.xsd 29

3 Locale Codes **35**

Language, Country, and Locale Codes 35

Index **38**

Preface

Welcome to the *Adobe® Device Central CS4 SDK Programmer's Reference*. This book describes the ExtendScript scripting API that allows you to create task plug-ins that extend the functionality of Adobe® Device Central's project feature.

About This Book

This book provides the reference information needed to create the files that make up an Adobe Device Central task plug-in. For conceptual information and examples, see the companion document, the *Adobe Device Central CS4 SDK Programmer's Guide*.

Who should read this book

This book is for developers who want to extend the capabilities of Adobe Device Central by creating their own task plug-ins. It assumes a general familiarity with the following:

- ExtendScript
- XML
- Any other Adobe Creative Suite® 4 applications you are using, such as Adobe Flash®

What is in this book

This book provides detailed reference information about Adobe Device Central. The information is presented in the following chapters:

Chapter 1, "Adobe Device Central DOM Object Reference," provides a complete API reference for the objects, properties, and functions defined in the Adobe Device Central document object model. These are the objects used in creating the `Task.jsx` file required by every Adobe Device Central task plug-in.

Chapter 2, "Device Central Task Schema," provides the XML schemas that define the allowable content in the Adobe Device Central task plug-in's XML file, `Task.xml`.

Chapter 3, "Locale Codes," lists the language/country combinations known as *locales* needed when organizing ZStrings for localization.

Adobe Device Central scripts have access to various utilities and tools that are part of ExtendScript, the Adobe extended implementation of JavaScript. These are described separately, in the *JavaScript Tools Guide*. This guide is available in the ExtendScript Toolkit application under **Help > SDK** or, alternatively, from `<DC_SDK>/Documents/JavaScript Tools Guide CS4.pdf` where `<DC_SDK>` is the SDK install location.

Document Conventions

Typographical conventions

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.

JavaScript conventions

This reference does not list properties and methods provided by the JavaScript language itself. For example, JavaScript objects commonly provide a `toString()` method and many objects in the Device Central SDK implement that method. However, this book does not describe such methods unless they differ from the standard JavaScript implementation.

Folders referenced in this document

The folders referenced in this document are summarized here. For convenience, they can 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 user tasks folder, 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

The default paths for the Adobe Device Central installation folder are listed in table below.

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 are given in the table below. You will 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

Operating System	Path
Mac OS	`/Applications/Utilities/Adobe Utilities/ExtendScript Toolkit CS4`

Operating System	Path
Windows XP	*system drive*\Program Files\Adobe\Adobe Utilities\ExtendScript Toolkit CS4
Windows Vista	*system drive*\Program Files\Adobe\Adobe Utilities\ExtendScript Toolkit CS4

Where to Go for More Information

The Adobe Device Central Software Developer's Kit (SDK) contains the ExtendScript documentation and code samples. The SDK is available for download from the Adobe website http://www.adobe.com/devnet/sdks.html. You can install the SDK in a folder with the name and location of your choice, referred to here as <DC_SDK>. The SDK contains these components.

<DC_SDK>/Documents/ADC SDK Guide.pdf	The programmer's guide
<DC_SDK>/Documents/ADC SDK Reference.pdf	This programmer's reference
<DC_SDK>/Documents/ JavaScript Tools Guuide CS4.pdf	*JavaScript Tools Guide*
<DC_SDK>/Documents/adat.xsd <DC_SDK>/Documents/task.xsd	XML schemas that describe the structure, content, and semantics of the XML document needed to define an Adobe Device Central task plug-in
<DC_SDK>/Sample Plugins/	A set of code samples that demonstrate Adobe Device Central scripting techniques
HelloWorld	A basic sample useful in getting started with the ADC SDK
CreateZip	A task plug-in that demonstrates the use of Adobe Device Central UI components
SendToWeb	A task plug-in that demonstrates how to send assets to an external service
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 for use with the Send to Web sample task plug-in

This book does not describe the JavaScript language. For 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

1 Adobe Device Central DOM Object Reference

This document provides a complete reference for the objects in the Adobe Device Central SDK document object model (DOM). See "Adobe Device Central Object Model" in the *Adobe Device Central CS4 SDK Programmer's Guide* for an overview of the object model and its use in creating plug-ins.

In addition to these Adobe Device Central-specific objects, your plug-ins can use ExtendScript, the Adobe extended implementation of JavaScript. The ExtendScript API is common to most JavaScript-enabled Adobe applications.

The following ExtendScript features are described in detail in the *JavaScript Tools Guide*:

- The ExtendScript `File` and `Folder` objects that provide portable access to the file system
- An interapplication messaging framework that provides the ability to communicate among scriptable applications using JavaScript
- ExtendScript utilities, including tools for debugging and for localization

Object Summary

The objects provided by the Adobe Device Central SDK are presented alphabetically. Complete syntax details are given for each object's constructor, properties, and functions.

Object	Description
Bluetooth	Manages interactions between local and remote Bluetooth® devices.
Control	Represents a distinct user interface widget used to display and retrieve information from the task's dialog.
Device	Provides access to device profile information.
ListItem	Manages the creation of utility objects that take dynamic properties.
Project	Exposes project-specific information.
StatusDialog	Informs the user of task execution progress and status.
Task	Represents an Adobe Device Central plug-in as a whole and provides access to its components.
URLStream	Provides interaction with external resources.
ZIPFile	Represents an archive file created with a plug-in.

Bluetooth

The Bluetooth class provides functionality that allows Device Central to connect from one Bluetooth-enabled device to other Bluetooth-enabled devices. For example, a Bluetooth object can be used to allow users to transfer files from a computer running Device Central to an external device.

Look in the `<ADC>/<REQUIRED_TASKS>` folder for the sample Bluetooth plug-in that ships with Adobe Device Central.

Bluetooth class methods

Method	Parameters	Returns	Description
`Bluetooth.cancel ()`	-	undefined	Cancels all activities. Halts any ongoing processes such those that are sending files or detecting external devices. No return value.
`Bluetooth.getDeviceAddressByIndex (index)`	`Number`	`String`	Returns the MAC address of the device at the specified index. **NOTE:** `index` is a zero-based index of the position of the device in the list of detected devices. The value of `index` must be smaller than the number of Bluetooth devices. If the value is out of range, this call returns an empty `String`.
`Bluetooth.getDeviceCount ()`	-	`Number`	Determines the number of Bluetooth-enabled devices that will be detected by a call to `Bluetooth.searchDevices()`. Returns the number of devices detected. **NOTE:** If you fail to search for devices before invoking this method, the count returned is `0`.

Method	Parameters	Returns	Description
`Bluetooth.getDeviceMajorClassByIndex (index)`	`Number`	`Number`	Determines the type of the device at the specified index. Returns a `Number` representing the major class of the device selected. See "Major classes" on page 12 for a description of each of the classes. **NOTE:** `index` is a zero-based index representing the position of the device in the list of detected devices. The value of index should be smaller than the device count detected. If the index is out of range this call returns `-1`.
`Bluetooth.getDeviceNameByIndex (index)`	`Number`	`String`	Obtains the name of the device at the specified `index`. Returns a string containing the device name, if set. **NOTE:** `index` is a zero-based index representing the position of the device in the list of detected devices. The value of index should be smaller than the device count detected. If the index is out of range this call returns an empty `String`.

Method	Parameters	Returns	Description
`Bluetooth.getStatus()`	-	`Number`	Obtains the status code for last send activity. Returns a number representing the status of last sending activity. The possible return values are listed in the table below. The default value is `0`. **Value** / **Description** `0` — Send successfully completed. `1` — Could not connect to device. `2` — Send canceled by device.
`Bluetooth.isAvailable()`	-	`Boolean`	Determines if the local machine is Bluetooth-enabled. Returns `true` if enabled; `false` otherwise. **NOTE:** Do not attempt to use Bluetooth functionality without first verifying that it is available.
`Bluetooth.isSearching()`	-	`Boolean`	Determines if detection of Bluetooth devices is ongoing. Use this call to determine if the search is ongoing so that you provide a user interface indicator such as a progress bar or icon. When the search is complete, you can display the list of devices returned. Returns `true` if an inquiry is underway; `false` otherwise.
`Bluetooth.isSending()`	-	`Boolean`	Determines if the Bluetooth library is in the process of sending a file. Returns `true` if the send is ongoing; `false` otherwise.

Method	Parameters	Returns	Description
`Bluetooth.searchDevices ()`	-	`Boolean`	Searches for external Bluetooth devices. Returns `true` if Bluetooth hardware is detected and enabled allowing a search for Bluetooth devices to be initiated; `false` otherwise.
`Bluetooth.sendFile (macAddress, file)`	`String` `String`	`Boolean`	Sends a file with the specified path to a the device with the specified address. Returns `true` if the send is successfully initiated; `false` otherwise. **NOTE:** `macAddress` is a device's unique 12-character hardware number in the notation `XX:XX:XX:XX:XX:XX`. `file` is the path of the file to send.

Major classes

The following table lists the major classes of Bluetooth devices. In this table, the **Value** column lists the possible values returned by the function `Bluetooth.getDeviceMajorClassByIndex()`. An unclassified device can have any number other than the values `0` to `6`.

Value	Description
`0`	Miscellaneous
`1`	Computer
`2`	Phone
`3`	LAN access point
`4`	Audio
`5`	Peripheral
`6`	Imaging
Other	Unclassified

Control

`Control` objects represent user interface widgets such as check boxes, text fields, and file lists. They are used to display information and to retrieve input from the user via a task configuration dialog.

`Control` objects must be created in the task XML file. Each control must be valid with respect to the Device Central Task Schema. New controls cannot be instantiated dynamically from the task's plug-in.

When defined in the XML file, controls can be accessed and updated via the task's plug-in file. A control can be accessed in the task plug-in by passing the control `id` to the task's getElement( ) method.

The `Task` object provides helper methods that work with specific types of controls. These methods make for more readable code. See "Task" for information on the available methods.

Control object in XML

The following example shows a `Control` object as it appears in the XML file:

```
<textField id="myTextFiled" width="150" sizex="scale"/>
```

Accessing a control object from a script

The examples that follow show alternative ways to access a `Control` object from a script:

```
task.getElement("myTextField")

task.getTextField("myTextField")
```

Control objects

Device Central provides the control types listed in the table below.

Object	Description
`button`	Small round clickable control used to invoke a script function.
`checkBox`	Dual-state control that displays a box with a checkmark when the value is `true`. The box is empty when the value is `false`.
`comboBox`	Clickable control that allows the user to select a single value from a drop-down or pop-up list.
`container`	Layout control used to group other controls in the configuration dialog.
`fileList`	Control that is similar to a `listBox`, but displays an iconic representation of a file.
`listBox`	Displays a list of the choices represented by the `ListItem` object. Users can select one or more options from the list.
`passwordField`	Single-line text input control used to obscure sensitive input.
`separator`	Layout control used to differentiate between sections in the user interface.
`staticText`	A text label that is not subject to user editing. Used to label controls or to otherwise provide information to the user.
`textField`	Single-line text input control subject to user editing.

Control properties

A control instance provides the properties listed in the table below.

Property	Type	Description
`checked`	`Boolean`	Toggle that is `true` if control is selected; `false` otherwise.
`enabled`	`Boolean`	Toggle that is `true` if the control accepts input; `false` otherwise.
`label`	`String`	Display text that is not user editable.
`list`	`ListItem Array`	Array of choice items displayed in the control.
`selectedItem`	`ListItem`	Currently selected item in a single-selection list.
`selectedItems`	`ListItem Array`	Read only. Array of selected items in a multi-select list.
`text`	`String`	Text displayed in the control's field.

Accessible properties by control object

The properties accessible from the task's script vary by type of `Control` object. A list of the properties available to each control appears in the table below.

	enabled	checked	label	selectedItem	selectedItems	list	text
button	✔		✔				
checkBox	✔	✔	✔				
comboBox	✔			✔		✔	
container	✔						
fileList	✔				✔	✔	
listBox	✔				✔	✔	
passwordField	✔						✔
separator							
staticText	✔		✔				
textField	✔						✔

Control object examples

The following examples illustrate the use of `Control` objects. The first demonstrates how to read a value while the second shows how to write a value.

```
//Get the input entered in a text field with id "myTextField"
var value = task.getTextField("myTextField").text;
```

```
//Mark to checked the check box control "myCheckBox")
task.getCheckBox("myCheckBox").checked = true;
```

Device

The `Device` object provides device-specific information. You can access the profile data for all of the devices in your project from the task script.

All device attributes can be accessed by parsing the profile XML file. The ExtendScript XML object API provides a simple way to work with XML data and to retrieve the information required. See the *JavaScript Tools Guide* for more information.

An example that illustrates how to work with a device profile is shown below:

```
var profileXML = new XML(device.profileXML);
var make = profileXML.coreData.manufacturer[0].@value
```

Device properties

Property	Type	Description
`name`	`String`	Read only. Device name.
`profileID`	`String`	Read only. The Device Central device identifier.
`profilePath`	`String`	Read only. The operating system-specific path to the device's XML profile file.
`profileXML`	`String`	Read only. The device's XML profile content.

ListItem

`ListItem` is a utility method that makes it possible to create objects with dynamic properties.

The `fileList`, `listBox`, and `comboBox` controls are populated with arrays of `ListItem`s. To successfully populate these controls, their `dataField` attributes must match the name of a `ListItem` dynamic property.

NOTE: `ListItem` must be instantiated. Its constructor does not take any arguments.

ListItem properties

Property	Type	Description
`selected`	`Boolean`	Indicator as to whether the item is selected within a `fileList`, `listBox`, or `comboBox`.

XML UI description for list box example

The following is an XML UI description for a list box that displays the name and value of a list of items.

```
<listBox id="properties">
    <columns>
        <column dataField="name" columnText="Name of the property" />
        <column dataField="value" columnText="Value of the property" />
    </columns>
</listBox>
```

Array of ListItems example

The ExtendScript code below creates an `Array` of `ListItem`s and populates each instance with the dynamic properties name and value expected in the XML.

When the array of items is created, you can use it to populate the `listBox` control defined in the XML as shown in the example below:

```
var properties = new Array;
for(var i = 0; i < 10; i++)
{
    var item = new ListItem;
    item.name = "Name " + i;
    item.value = i;
    properties.push(item);
}
task.getElement("properties").list = properties;
```

Project

The `Project` object represents an Adobe Device Central project. Each task has access to an instance of the project the task is running within. This object provides project information and a way to access and modify project resources and devices dynamically.

NOTE: As the `Project` object does not provide a constructor, new project instances cannot be created. However, a `project` variable that represents an instance of the current project is always available to the task script.

Project properties

Property	Type	Description
devices	Array of Device	Read only. An array of all the project's devices.
name	String	Read only. The name of the project .
path	String	Read only. The project's file system location. Returns null if project was not saved .
resources	Array of String	Read only. The file system path to all the resources available in the project.
selectedDevices	Array of Device	Read only. An array with all of the project's currently selected devices.
selectedResources	Array of String	Read only. The file system path to all the currently selected project resources.

Project methods

Method	Parameters	Returns	Description
`addFolder( path);`	`String`	`Boolean`	Adds the specified folder hierarchy to the resources section of the project, allowing for better organization of the project's resources folder. Returns `true` if the folder was successfully added to the project; `false` otherwise. **NOTE:** Folder names are separated by a slash (/). For example, `folder1/childFolder/childChildFolder`.
`addResource( file, path)`	`String` `String`	`Boolean`	Adds the specified file to the project. If a project path is provided, the resource is stored under that folder hierarchy. Returns `true` if the file was successfully added to the project, `false` otherwise. **NOTE:** `false` is returned if the file specified already exists in the project. **NOTE:** `Path` is not the path to the file, but to the folder hierarchy where the file should be stored. **NOTE:** Folder names are separated by a slash (`/`). For example, `folder1/childFolder/childChildFolder`.

Method	Parameters	Returns	Description
`getResourcesFromFolder( path);`	`String`	`Array` of `String`	Returns an array of `String` paths representing the locations of the files found in the project folder hierarchy under the path specified. **NOTE:** `Path` is the folder hierarchy where the files are found. **NOTE:** Folder names are separated by a slash (/). For example, `folder1/childFolder/childChildFolder`.
`openResourcesDialog ();`	-	`Array` of `String`	Invokes a resources dialog that allows the user to select resource files or browse the machine's file system. Returns an array of `String` paths to the files selected in the dialog. **NOTE:** The user sees only those files that have already been added to the project.

StatusDialog

The `StatusDialog` represents a user-interface component that informs the user of task execution progress and status. It is accessed through the `Task` object.

A `StatusDialog` should be displayed during task execution. This gives the user gets visual feedback on a task's progress and allows for notification of any errors during the task execution.

`StatusDialog` allows developers to customize the title of the dialog box and the text displayed. The **Done** and **Cancel** buttons also can be customized to point to a specific script function that can be enabled or disabled as required.

Note: The `StatusDialog` object does not provide a constructor. New instances of a `StatusDialog` cannot be created.

StatusDialog properties

Property	Type	Description
`text`	`String`	Status dialog progress text
`title`	`String`	Status dialog title

StatusDialog methods

Method	Parameter type	Returns	Description
`enableCancelButton( status)`	`Boolean`	undefined	Sets the enable state of the **Cancel** button in the status dialog. Set to `true` to enable the button, `false` to disable it.
`enableDoneButton( status)`	`Boolean`	undefined	Sets the enable state of the **Done** button in the status dialog. Set to `true` to enable the button, `false` to disable it.
`end ()`	-	undefined	Closes the status dialog.
`modalDuring ()`	-	undefined	Dispatches OS events while a modal status dialog is displayed.
`start (cancelFunction, doneFunction)`	`String` `String`	undefined	Invokes the status dialog. **NOTE:** `cancelFunction` is a `String` that represents the function to be called when the user clicks the **Cancel** button in the dialog. `doneFunction` is a `String` that represents the function to be called when the user clicks the **Done** button in the dialog.

Task

A `Task` object represents a Device Central plug-in. A single global instance of this object is created each time a task is created. Access the object using the `task` global variable. The `Task` object provides access to the user-interface controls in the configuration dialog and also allows the invocation of the status and prompt dialogs. Use the `Task` object to also interact with external applications or to query the environment in which Device Central is running.

NOTE: The `Task` object does not provide a constructor and therefore new instances of a `Task` cannot be created.

Task properties

Property	Type	Description
`activeDevice`	`Device`	Read only. Returns the `Device` object that is currently available in the Adobe Device Central emulator. If no device is currently loaded, it returns `null`.
`isOSMacintosh`	`Boolean`	Read only. Indicator as to whether a script is running on a Macintosh. Has the value `true` if it is; `false` otherwise.

Property	Type	Description
isOSWindows	Boolean	Read only. Indicates whether a script is running on Windows. Return true if it is; false otherwise.
scriptLocation	String	Read only. The absolute OS-specific path to the task.
statusDialog	StatusDialog	Read only. The status dialog object.

Task methods

Method	Parameters	Returns	Description
executeBlocking(pathToApplication, arguments);	String Array	undefined	Starts an external application with a variable number of command line arguments. Returns when the external application is exited. **NOTE:** pathToApplication is the absolute path to the application to be executed. arguments is an Array of command line parameters. For example, task.executeBlocking("/Application/TextEdit ",["param1", "param2"]);
executeNonBlocking(pathToApplication, arguments);	String Array	undefined	Starts an external application with a variable number of command line arguments. Returns immediately when an external application is launched. **NOTE:** pathToApplication is the absolute path to the application to be executed. arguments is an Array of command line parameters. For example, task.executeBlocking("/Application/TextEdit ",["param1", "param2"]);
getButton (id);	String	Control	Returns a control object of type button if found; null if not found. **NOTE:** id is a unique identifier for the control specified in the XML user interface description file.

Method	Parameters	Returns	Description
`getCheckBox (id);`	`String`	`Control`	Returns a control object of type `checkBox` if found; `null` if not found. **NOTE:** `id` is a unique identifier for the control specified in the XML user interface description file.
`getComboBox (id);`	`String`	`Control`	Returns a control object of type `comboBox` if found; `null` if not found. **NOTE:** `id` is a unique identifier for the control specified in the XML user interface description file.
`getElement (id)`	`String`	`Control`	Returns a control object of any type if found; `null` if not found. **NOTE:** `id` is a unique identifier for the control specified in the XML user interface description file.
`getFileList (id)`	`String`	`Control`	Returns a control object of type `fileList` if found; `null` if not found. **NOTE:** `id` is a unique identifier for the control specified in the XML user interface description file.
`getListBox (id);`	`String`	`Control`	Returns a control object of type `listBox` if found; `null` if not found. **NOTE:** `id` is a unique identifier for the control specified in the XML user interface description file.
`getPasswordField (id);`	`String`	`Control`	Returns a control object of type `passwordField` if found; `null` if not found. **NOTE:** `id` is a unique identifier for the control specified in the XML user interface description file.
`getStaticText (id);`	`String`	`Control`	Returns a control object of type `staticText` if found; `null` if not found. **NOTE:** `id` is a unique identifier for the control specified in the XML user interface description file.

Method	Parameters	Returns	Description
`getTextField (id);`	`String`	`Control`	Returns a control object of type `textField` if found; `null` if not found. **NOTE:** `id` is a unique identifier for the control specified in the XML user interface description file.

`promptDialog( title, label, password );`	`String`
	`String`
	`Boolean`
		`String`	Invokes a prompt dialog box that waits for the user to enter text.
			
			Returns the entered text as a `String` or returns `null` if the user clicks the **Cancel** button.
			
			NOTE: `title` is the title of the dialog window.
			
			`label` is the text label for the dialog input box.
			
			If `password` is set to `true`, the text field obscures the user typing .

URLStream

The `URLStream` object provides the basic functionality needed to connect from a Device Central task to a remote server over a TCP connection. This object allows the connection and exchange of data with any server on the Internet, using protocols such as HTTP and FTP. This capability enables a task to retrieve data from an online service, post data to an external web application, or send files to an FTP server.

`URLStream` provides the calls `open()` and `close()` to establish or to terminate a connection, and `sendFile()` to transfer resources.

Using the `URLStream` object's properties, you can customize the connection depending on your server's requirements. For example, you can set up connections that access servers via a proxy, using authentication, or certificates.

After executing a connection, the variable `result` contains the response received from the server.

NOTE: `URLStream` must be instantiated. Its constructor takes no arguments.

Sending a request and reading the reply

In the following example, an instance of `URLStream` connects to an HTTP server (which listens on port 80); it then sends a very simple HTTP `GET` request to obtain the contents of the site specified, and then it reads the reply, which is the HTML of the home page.

```
var stream = new URLStream;
stream.url = "http://www.adobe.com";
stream.port = 80
stream.open();
while(stream.isConnected) {
```

```
    //show status dialog while waiting for response
}
var response = stream.result;
```

For more details on how to use `URLStream`, see the code example in the SDK sample plug-in `SendToWeb`. See the *SendToWeb README* file for more information.

See the `SendToFTP` task that ships with Adobe Device Central for an example on how to implement an FTP service. It is located in the `<REQUIRED_TASKS>` folder.

URLStream properties

Properties	Type	Description
`additionalHeader`	`String`	Additional header data.
`authentication`	`String`	Sets the authorization policy header; possible values are `none`, `basic`, `digest`, `gssnegotiate`, `ntlm`, `any`, `anysafe`.
`certificateFile`	`String`	Absolute location of the certification file.
`certificateFormat`	`String`	Format of the certificate file, either `PEM` or `DER`. `PEM` encoding is the default.
`contentType`	`String`	Header field that indicates the type of data sent to the server, for example, `text/html`, `multipart/form-data`.
`error`	`Number`	Read only. Error code identifying a problem that has occurred. See URLStream error codes for a list of the possibilities.
`followRedirect`	`Boolean`	Toggle indicating whether or not redirects should be followed. If `true`, a request redirects; if `false`, it does not.
`httpStatusCode`	`Number`	HTTP status code returned by the server when performing an HTTP request. A successful request returns `200`. Visit the following W3C website for more information: `http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`
`isConnected`	`Boolean`	Read only. Toggle indicating whether or not a connection is active. A return value of `true` indicates that it is; `false` that the connection is lost.
`password`	`String`	Password, if authentication is required.
`port`	`Number`	The port number to connect to.

Properties	Type	Description
postData	String	Data to post as expected by the external server. For example, the code `stream.postData = "a=1&b=2";` sets parameters `a` and `b` to `1` and `2`, respectively.
proxyAuthentication	String	Sets the proxy authorization policy header; possible values are: `none`, `basic`, `digest`, `gssnegotiate`, `ntlm`, `any`, `anysafe`.
proxyName	String	The URL address of the proxy server to connect to.
proxyPassword	String	Proxy server password, if authentication is required.
proxyPort	Number	The port number of the proxy server to connect to.
proxyUsername	String	Proxy server username, if authentication is required.
result	String	Read only. Result of a request.
unrestrictedAuthorization	Boolean	Toggle indicating whether to continue to send authentication information (username and password) when being redirected, even in cases where the hostname has changed. Only meaningful when `followRedirect` is set to `true`.
url	String	A `String` representing a URL's connection address.
username	String	The server `username`, if authentication is required.
verifyHost	Boolean	Toggle indicating whether the library verifies that the server certificate is for the expected server. The value is `true` if it does; `false` otherwise.
verifyPeer	Boolean	Toggle that when `true` verifies the authenticity of the peers certificate; does not verify if `false`.

URLStream methods

Method	Parameters	Returns	Description
`cancel()`	-	`undefined`	Cancels the current connection.
`open()`	-	`undefined`	Opens a network connection with the specified attributes. When using the HTTP protocol, `open()` makes a `GET` request to the server. If `postData` is available `URLStream` will `POST` the request.
`sendFile( path)`	`String`	`undefined`	Sends a file over the network to the specified URL. There's no need to invoke `open()` when using `sendFile()`. **NOTE:** `path` is a string representing the location of the file to be sent.

URLStream error codes

The following table lists the error codes returned by the variable `error`.

Value	Description
0	OK
1	UNSUPPORTED_PROTOCOL
2	URL_MALFORMAT
3	COULDNT_RESOLVE_PROXY
4	COULDNT_RESOLVE_HOST
5	COULDNT_CONNECT
6	FTP_ACCESS_DENIED
7	HTTP_RETURNED_ERROR
8	WRITE_ERROR
9	READ_ERROR
10	OPERATION_TIMEOUTED
11	HTTP_POST_ERROR
12	SSL_CONNECT_ERROR
13	TOO_MANY_REDIRECTS
14	SSL_PEER_CERTIFICATE

Value	Description
15	SSL_ENGINE_NOTFOUND
16	SSL_ENGINE_SETFAILED
17	SSL_CERTPROBLEM
18	LOGIN_DENIED
19	UNKNOWN

ZIPFile

A `ZIPFile` object represents a Zip archive, a file that contains one or more compressed files. The `ZIPFile` object API allows you to create Zip archives from your task and add files to it.

NOTE: `ZIPFile` must be instantiated. Its constructor does not take any arguments.

The `CreateZip` SDK samples shows how to create an archive file using the `ZipFile` object.

ZipFile methods

Method	Parameters	Returns	Description

`addFile(`
`  filename,`
`  path,`
`  comment`
`);`
	
	`String`
	`String`
	`String`
		`Boolean`	Adds a file to the Zip archive created.
			
			Returns `true` if the file was successfully added to the archive; `false` otherwise.
			
			➤ `filename` is the name to be used in the Zip archive for the file added.
			
			➤ `path` is a `String` representing the file to be added.
			
			➤ `comment` is a text remark to be associated with the file being added.
			
			NOTE: Provide a folder path if a folder hierarchy is required when the archive is extracted. If a flatter structure is preferable, you need only provide the file name. For example, `/assets/myasset.png` creates an assets folder when extracted and `myasset.png` is found under the `assets` folder.

Method	Parameters	Returns	Description
`close ();`	-	undefined	Closes the Zip archive.
`open (path);`	`String`	`Boolean`	Opens a Zip archive at the specified location. If a file exists with the same name it is overwritten. If no archive exists, a new one is created. Returns `true` if the archive was successfully opened, `false` otherwise. **NOTE:** The `path` parameter states an OS specific path where the archive is to be saved. The path must include the name of the file and the desired extension. For example, `C:\test\myarchive.zip`.

2 Device Central Task Schema

The files `adat.xsd` and `task.xsd` together constitute an XML schema definition that describes the required structure for the `task.xml` file required for every Device Central plug-in.

- adat.xsd describes the high-level structure of `task.xml`. It describes the highest level element task and imports the schema `task.xsd`.
- task.xsd defines the elements referenced by the highest level element `task`.

NOTE: These files are provided as part of the Adobe Device Central SDK.

adat.xsd

The `adat.xsd` file is an XML schema that governs the allowable content for the file `task.xml`. In particular, the file defines the content required by the element `task`. This file imports the `task.xsd` schema file.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:adat="http://ns.adobe.com/devicecentral/task/"
       targetNamespace="http://ns.adobe.com/devicecentral/task/"
       elementFormDefault="qualified">
   <xs:import schemaLocation="Task.xsd"/>
   <xs:element name="task">
       <xs:complexType>
           <xs:sequence>
               <xs:element ref="general"/>
               <xs:element ref="dialog"/>
           </xs:sequence>
           <xs:attribute name="guid" type="xs:string" use="required"/>
           <xs:attribute name="version" type="xs:integer" use="required"/>
       </xs:complexType>
   </xs:element>
</xs:schema>
```

task.xsd

The file `task.xsd` is an XML schema that describes the allowable content for the elements `general` and `dialog` referenced in `adat.xsd`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:adat="http://ns.adobe.com/devicecentral/task/"
       elementFormDefault="qualified">
   <xs:import namespace="http://ns.adobe.com/devicecentral/task/"
          schemaLocation="adat.xsd"/>
   <!-- info section -->
   <!-- general -->
   <xs:element name="general">
       <xs:complexType>
           <xs:all>
               <xs:element ref="title"/>
               <xs:element ref="menuItem"/>
               <xs:element ref="toolTip"/>
```

```
            <xs:element ref="script"/>
            <xs:element ref="icon"/>
            <xs:element ref="toolbar"/>
            <xs:element ref="localization" minOccurs="0"/>
            <xs:element ref="libraries" minOccurs="0"/>
        </xs:all>
    </xs:complexType>
</xs:element>
<!-- title -->
<xs:element name="title">
    <xs:complexType>
        <xs:attribute name="value" use="required"/>
    </xs:complexType>
</xs:element>
<!-- menuItem -->
<xs:element name="menuItem">
    <xs:complexType>
        <xs:attribute name="value" use="required"/>
    </xs:complexType>
</xs:element>
<!-- toolTip -->
<xs:element name="toolTip">
    <xs:complexType>
        <xs:attribute name="value" use="required"/>
    </xs:complexType>
</xs:element>
<!-- script -->
<xs:element name="script">
    <xs:complexType>
        <xs:attribute name="path" type="xs:NCName" use="required"/>
    </xs:complexType>
</xs:element>
<!-- icon -->
<xs:element name="icon">
    <xs:complexType>
        <xs:attribute name="path" type="xs:NCName" use="required"/>
    </xs:complexType>
</xs:element>
<!-- localization -->
<xs:element name="localization">
    <xs:complexType>
        <xs:attribute name="path" type="xs:NCName" use="required"/>
    </xs:complexType>
</xs:element>
<!-- libraries -->
<xs:element name="libraries">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="library"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<!-- library -->
<xs:element name="library">
    <xs:complexType>
        <xs:attribute name="name" type="xs:NCName" use="required"/>
    </xs:complexType>
</xs:element>
<!-- toolbar -->
<xs:element name="toolbar">
```

```
    <xs:complexType>
        <xs:attribute name="value" type="xs:boolean" use="required"/>
    </xs:complexType>
</xs:element>
<!-- UI description -->
<!-- dialog -->
<xs:element name="dialog">
    <xs:complexType>
        <xs:sequence>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element ref="listBox"/>
                <xs:element ref="fileList"/>
                <xs:element ref="container"/>
                <xs:element ref="staticText"/>
                <xs:element ref="textField"/>
                <xs:element ref="passwordField"/>
                <xs:element ref="button"/>
                <xs:element ref="separator"/>
                <xs:element ref="checkBox"/>
                <xs:element ref="comboBox"/>
            </xs:choice>
        </xs:sequence>
        <xs:attribute name="height" type="xs:integer" use="required"/>
        <xs:attribute name="minHeight" type="xs:integer" use="required"/>
        <xs:attribute name="minWidth" type="xs:integer" use="required"/>
        <xs:attribute name="width" type="xs:integer" use="required"/>
    </xs:complexType>
</xs:element>
<!-- container -->
<xs:element name="container">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="baseType">
                <xs:sequence>
                    <xs:choice minOccurs="0" maxOccurs="unbounded">
                        <xs:element ref="listBox"/>
                        <xs:element ref="fileList"/>
                        <xs:element ref="container"/>
                        <xs:element ref="staticText"/>
                        <xs:element ref="textField"/>
                        <xs:element ref="passwordField"/>
                        <xs:element ref="button"/>
                        <xs:element ref="separator"/>
                        <xs:element ref="checkBox"/>
                        <xs:element ref="comboBox"/>
                    </xs:choice>
                </xs:sequence>
                <xs:attribute name="layout" type="layoutType"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!--  listBox -->
<xs:element name="listBox">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="listType">
                <xs:attribute name="selectable" type="xs:boolean"/>
            </xs:extension>
        </xs:complexContent>
```

```
    </xs:complexType>
</xs:element>
<!-- fileList -->
<xs:element name="fileList">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="listType">
                <xs:attribute name="icon" type="xs:NCName"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!-- checkBox -->
<xs:element name="checkBox">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="buttonType">
                <xs:attribute name="checked" type="xs:boolean"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!-- comboBox -->
<xs:element name="comboBox">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="clickable">
                <xs:attribute name="dataField" type="xs:NCName"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!-- separator -->
<xs:element name="separator">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="baseType"/>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!-- staticText -->
<xs:element name="staticText">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="baseType">
                <xs:sequence>
                    <xs:element name="label">
                        <xs:complexType>
                            <xs:attribute name="value" use="required"/>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
                <xs:attribute name="alignment" type="xs:NCName"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!-- textField -->
<xs:element name="textField">
    <xs:complexType>
```

```
        <xs:complexContent>
            <xs:extension base="baseType">
                <xs:sequence>
                    <xs:element name="text" minOccurs="0">
                        <xs:complexType>
                            <xs:attribute name="value" type="xs:integer"
                                    use="required"/>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!-- passwordField -->
<xs:element name="passwordField">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="baseType">
                <xs:sequence>
                    <xs:element name="text" minOccurs="0">
                        <xs:complexType>
                            <xs:attribute name="value" type="xs:integer"
                                    use="required"/>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!-- button -->
<xs:element name="button">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="buttonType"/>
        </xs:complexContent>
    </xs:complexType>
</xs:element>
<!-- buttonType -->
<xs:complexType name="buttonType">
    <xs:complexContent>
        <xs:extension base="clickable">
            <xs:sequence>
                <xs:element name="label" minOccurs="0">
                    <xs:complexType>
                        <xs:attribute name="value" type="xs:string" use="required"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- Clickable -->
<xs:complexType name="clickable">
    <xs:complexContent>
        <xs:extension base="baseType">
            <xs:attribute name="onClick" type="xs:NCName"/>
        </xs:extension>
    </xs:complexContent>
```

```
    </xs:complexType>
    <!--  listType -->
    <xs:complexType name="listType">
        <xs:complexContent>
            <xs:extension base="baseType">
                <xs:sequence>
                    <xs:element ref="columns"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- columns -->
    <xs:element name="columns">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="column" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <!-- column -->
    <xs:element name="column">
        <xs:complexType>
            <xs:attribute name="columnText" use="required"/>
            <xs:attribute name="dataField" type="xs:NCName" use="required"/>
            <xs:attribute name="width" type="xs:integer"/>
        </xs:complexType>
    </xs:element>
    <!-- baseType -->
    <xs:complexType name="baseType">
        <xs:attribute name="id" type="xs:NCName"/>
        <xs:attribute name="sizex" type="sizexType"/>
        <xs:attribute name="width" type="xs:integer"/>
        <xs:attribute name="height" type="xs:integer"/>
        <xs:attribute name="enabled" type="xs:boolean"/>
    </xs:complexType>
    <!-- sizexType -->
    <xs:simpleType name="sizexType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="left"/>
            <xs:enumeration value="scale"/>
            <xs:enumeration value="right"/>
        </xs:restriction>
    </xs:simpleType>
    <!-- layoutType -->
    <xs:simpleType name="layoutType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="vertical"/>
            <xs:enumeration value="horizontal"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

3 Locale Codes

The table that follows lists ISO codes for language/country combinations known as *locales*.

The lists contains all the Adobe supported language/country combinations suitable for display in Adobe Device Central user interfaces. In cases where Adobe does not support a given code, your program should default to a supported code. For example, `de_CH` (German, Switzerland) is not currently supported and should default to `de_DE` (German, Germany).

Languages are listed in alphabetical order with variants grouped together in sorted order by country of use. The following information is provided:

- *Language* is the English language name for the language in question.
- *Country* is the country in which the language is spoken.
- *Code* is the name to use for the directory that holds that country's ZStrings.

  See "Using ZStrings for Localization" in the *Adobe Device Central SDK Programmer's Guide* for more information.

Asian language names are expressed using Macintosh native system script encodings. On Windows, the language names only appear if the installer used is a multi-language installer. In such cases, the installer shows a **Choose Setup Language** dialog and the names come from the InstallShield file, `lang.dat`.

Language, Country, and Locale Codes

Language	Country	Locale Code
Albanian	Albania	`sq_AL`
Arabic	Saudi Arabia	`ar_SA`
Arabic	United Arab Emirates	`ar_AE`
Azerbaijani	Azerbaijan	`az`
Azerbaijani, Azerbaijan	Azerbaijan	`az_Cyrl_AZ`
Azerbaijani, Azerbaijan	Azerbaijan	`az_Arab_AZ`
Belarusian	Belarus	`be_BY`
Bulgarian	Bulgaria	`bg_BG`
Catalan	Spain	`ca_ES`
Chinese	-	`zh`
Chinese, Simplified	China	`zh_CN`
Chinese, Simplified	Singapore	`zh_SG`
Chinese, Traditional	Hong Kong	`zh_HK`

Language	Country	Locale Code
Chinese, Traditional	Macau	zh_MO
Chinese, Traditional	China, Taiwan	zh_TW
Croatian	Croatia	hr_HR
Czech	Czech Republic	cs_CZ
Danish	Denmark	da_DK
Dutch	Netherlands	nl_NL
English, International	-	en
English, International	World excluding NAFTA	en_GB
English, U.S.	U.S.A.	en_US
Estonian	Estonia	et_EE
Finnish	Finland	fi_FI
French	-	fr
French	Canada	fr_CA
French	France	fr_FR
French	Middle East	fr_XM
German	-	de
German	Germany	de_DE
German, Reformed spelling	Germany	de_DE_1996
German, Old spelling	Germany	de_DE_1901
Greek	Greece	el_GR
Hebrew	Israel	he_IL
Hungarian	Hungary	hu_HU
Hindi	India	hi_IN
Icelandic	Iceland	is_IS
Italian	Italy	it_IT
Japanese	Japan	ja_JP
Kazakh	Kazakhstan	kk_KZ
Korean	Korea	ko_KR
Latvian	Latvia	lv_LV
Macedonian	Macedonia	mk_MK
New Norwegian	Norway	nn_NO

Language	Country	Locale Code
Norwegian	Norway	nb_NO
Polish	Poland	pl_PL
Portuguese	-	pt
Portuguese, Brazilian	Brazil	pt_BR
Portuguese, Portugal	Portugal	pt_PT
Romanian	Romania	ro_RO
Russian	Russia	ru_RU
Serbian	-	sr
Serbian, Serbia & Montenegro	Serbia & Montenegro	sr_Cyrl_CS
Serbian, Serbia & Montenegro	Serbia & Montenegro	sr_Latn_CS
Serbo-Croatian	Yugoslavia	sh_YU
Slovak	Slovakia	sk_SK
Slovenian	Slovenia	sl_SI
Spanish	-	sl_SI
Spanish	Spain	es_ES
Spanish, Latin America	Latin America	es_MX
Spanish, Mexico	Mexico	es_MX
Swedish	Sweden	sv_SE
Thai	Thailand	th_TH
Turkish	Turkey	tr_TR
Ukrainian	Ukraine	uk_UA
Vietnamese	Vietnam	vi_VN

Index

A

adat.xsd file, 29
ADC folder, 5
Asian language names, 35

B

Bluetooth
    class description, 8
    class methods, 9–12
    major device classes, 12

C

Control objects
    about, 13
    accessible properties, 14
    examples, 15
    in XML, 13
    provided properties, 14
    summary of, 13
conventions, typographical, 5
country locale codes, 35–37

D

DC_SDK folder, 5
Device object
    about, 15
    properties, 15
DOM object reference, 8

E

errors, URLStream, 26
ESTK folder, 5
examples
    Control objects, 15
    reference materials, 7

F

folders
    installation locations, 6
    summary, 5

I

ISO codes, 35–37

J

JavaScript
    conventions, 5
    references, 7

L

language locale codes, 35–37
ListItem object
    about, 15
    examples, 16
    properties, 16
locales
    codes, 35–37
    defaults, 35

O

objects
    Bluetooth, 8
    Control, 13
    Device, 15
    ListItem, 15
    Project, 16
    reference, 8
    StatusDialog, 19
    summary, 8
    Task, 20
    URLStream, 23
    ZIPFile, 27

P

Project object
    about, 16
    methods, 18
    properties, 17

R

reference materials, list of, 7
REQUIRED_TASKS folder, 5

S

StatusDialog object
    about, 19
    methods, 20
    properties, 19

T

Task object
    about, 20
    methods, 21–23
    properties, 20
task.xsd file, 29–34
TASKS folder, 5
typographical conventions, 5

U

URLStream object
    about, 23
    error codes, 26
    example, 23
    methods, 26
    properties, 24

Z

ZIPFile object
    about, 27
    methods, 27
ZStrings, locale codes, 35–37