

# Adobe Acrobat 7.0.5 SDK



## Snippet Runner Cookbook

November 8, 2005



Adobe Solutions Network — <http://partners.adobe.com>

Copyright 2005 Adobe Systems Incorporated. All rights reserved.

NOTICE: All information contained herein is the property of Adobe Systems Incorporated. No part of this publication (whether in hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the Adobe Systems Incorporated.

PostScript is a registered trademark of Adobe Systems Incorporated. All instances of the name PostScript in the text are references to the PostScript language as defined by Adobe Systems Incorporated unless otherwise stated. The name PostScript also is used as a product trademark for Adobe Systems' implementation of the PostScript language interpreter.

Except as otherwise stated, any reference to a "PostScript printing device," "PostScript display device," or similar item refers to a printing device, display device or item (respectively) that contains PostScript technology created or licensed by Adobe Systems Incorporated and not to devices or items that purport to be merely compatible with the PostScript language.

Adobe, the Adobe logo, Acrobat, the Acrobat logo, Acrobat Capture, Distiller, PostScript, the PostScript logo and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Apple, Macintosh, and Power Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. PowerPC is a registered trademark of IBM Corporation in the United States. ActiveX, Microsoft, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Verity is a registered trademark of Verity, Incorporated. UNIX is a registered trademark of The Open Group. Verity is a trademark of Verity, Inc. Lextek is a trademark of Lextek International. All other trademarks are the property of their respective owners.

This publication and the information herein is furnished AS IS, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied, or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes, and noninfringement of third party rights.



---

## Introduction

The SnippetRunner allows developers to quickly prototype code containing Acrobat or PDF Library API calls without the overhead of writing and verifying a complete plug-in or application. It provides an infrastructure and utility functions to support execution and testing of code snippets, which are small but complete portions of Acrobat plug-in or PDF Library application code.

The SnippetRunner architecture consists of these major components:

- A back-end server that provides the basic functionality, which includes a parameter input mechanism, debug support, and exception handling:
  - Acrobat SDK: The SnippetRunnerServer Acrobat plug-in.
  - PDF Library: The SnippetRunner application.
- A *Common User Interface* that acts as a client to the back-end server and provides Acrobat and PDF Library developers with a single cross-platform GUI.

This document contains the following sections:

- [“Installing and Running SnippetRunner” on page 4](#)
- [“Using the Common Interface” on page 7](#)
- [“Understanding and Writing Snippets” on page 12](#)

Where appropriate, this document points out differences in usage of SnippetRunner in the Acrobat SDK vs. the PDF Library SDK.

SnippetRunner is accompanied by more than 100 sample code snippets that demonstrate Acrobat and PDF Library API methods. The following documents list these snippets:

- *Guide to SDK Samples* for the Acrobat SDK.
- *PDF Library Overview* for the PDF Library SDK.

In addition, these Acrobat and PDF Library SDK documents are useful in writing and understanding snippets:

- The *Acrobat and PDF Library API Overview* and *Acrobat and PDF Library API Reference* describe the APIs you use in developing plug-ins and PDF Library applications.
- The *Acrobat Plug-in Guide* gives additional information about plug-in components, such as exception handlers.
- *Developing for Adobe Reader* provides an introduction to those portions of the Adobe Acrobat Software Development Kit (SDK) that pertain to your development efforts for Adobe Reader.

---

## Installing and Running SnippetRunner

### Building SnippetRunner

SnippetRunner consists of a set of files, organized into folders in the `Samples\SnippetRunner\` directory of the Acrobat SDK or PDF Library SDK. The files consist of the following:

- Source code files for the SnippetRunner server.

These files are located in the `Sources\platform\` directory. For each platform, there is a project file you can use to compile SnippetRunner along with the SDK header files. It consists of a `.mcp` file for Mac OS, a `.vcproj` file for Windows and makefiles for UNIX platforms.

Before using the SnippetRunner, you need to build it in the appropriate manner for your platform:

- For the Acrobat SDK, after the SnippetRunner project is built, the SnippetRunner Server plug-in will be installed in the appropriate directory so that it will be loaded by Acrobat when it is launched.
- For PDF Library, the SnippetRunnerServer application will be executed when the Common Interface is launched (see [“Starting the Common Interface” on page 5](#)).

- SnippetRunner environment and utility files.

These files are located in the `Sources\platform\Acrobat` (for the Acrobat SDK only) and `Sources\platform\Shared` directories.

- Individual code snippets.

Each of these is intended to demonstrate one or more APIs. Each snippet exists as a single, separate file within the `Sources\snippets\Acrobat` (for the Acrobat SDK only) or `Sources\snippets\Shared` directory, and is included in the SnippetRunner project.

If you build your own snippets (see [“Understanding and Writing Snippets” on page 12](#)), you can add them to the SnippetRunner project and rebuild the project.

- External files required by snippets.

These are in the `ExampleFiles` directory and can be sample files for input or resources for user interface components.

### Installing the Java Runtime Environment

The Common SnippetRunner Interface is a Java application that requires a platform-specific Java Virtual Machine (JVM) to be properly installed and accessible. The recommended JVM is the Java Runtime Environment (JRE) available for download from Sun Microsystems, Inc.

If you already have a JDK installed, you do not need a separate JVM for the Common SnippetRunner Interface. If you are specifically installing one for this application, J2SE 1.4.2

(or 1.5) JRE is recommended. Please follow the installation instructions provided by Sun to properly install the JRE.

## Starting the Server

The SnippetRunner back-end server must be started prior to starting the Common Interface front end to ensure the establishment of socket communication channels (see below for more information).

When running snippets as Acrobat plug-ins, you must launch the Acrobat viewer before starting the Common Interface. As long as you have copied the plug-in file SnippetRunnerServer plug-in to the Acrobat viewer's `plug_ins` directory, the SnippetRunner server loads on viewer launch and starts listening in on a port designated for socket connection requests.

When running snippets using the PDF Library SDK, you can start up the Common Interface (see next section). This automatically starts the PDF Library SnippetRunner application prior to attempting to establish a socket communication channel, so no manual invocation is required.

## Starting the Common Interface

You may load the Common Interface as a standalone Java application or as a Java applet. The following sections describe the procedures for each of these cases.

The Common Interface is packaged as a Java Archive (JAR) file containing Java byte code (class files) and associated resources. This file is `CommonInterface.jar` in the SnippetRunner folder. The JAR format provides for resource integrity and also allows the content of the archive to be digitally signed, which is required so that the Common Interface can be certified to run as an applet by a browser's JVM.

### Configuration File

The first time the Common Interface is invoked, a configuration file (`acrosdk.config` for the Acrobat SDK and `pdfsdk.config` for the PDF Library) is automatically generated and stored in the user's home directory.

To ensure that the contents of this file are created accurately, you must launch the Common Interface from its installed location in the SnippetRunner folder so that a "base" directory (a platform-specific absolute path to the SnippetRunner folder) can be properly written to the file. This path is used by the Common Interface to locate reference files and snippet source code at run time. Once this configuration file is written, the Common Interface can be invoked from any folder location.

If a path other than that of the SnippetRunner folder is written to the configuration file, the Common Interface will not function properly. If this occurs, delete the configuration file and restart the Common Interface from its base directory.

### Running as a Standalone Java Application

To run the client as a standalone Java application, follow these steps:

1. (*Acrobat SDK only*) Launch the Acrobat viewer.
2. Open a terminal/console window.
3. Switch to the directory where the `CommonInterface.jar` file resides (the SnippetRunner directory).
4. Execute the following command: `java -jar -cp . CommonInterface.jar`  
On Windows and Mac, as an alternative, double-click the `CommonInterface.jar` icon to launch the Common Interface. (This process assumes that the "JAR" file extension has not been associated with other applications after your JDK/JRE installation.)

In a short time, the Common SnippetRunner Interface should begin running.

### Running as a Java Applet

To run the Common Interface client as a Java applet, you must digitally sign the JAR file before loading the client into your browser. This requires the `keytool` and `jarsigner` command-line utilities from the J2SE Development Kit (or equivalent).

Follow these steps to sign the `CommonInterface.jar` file for running as an applet:

1. Generate a public/private key pair and the self-signed certificate.

Issue a command similar to this from a console using the `keytool` utility:

```
console>keytool -genkey -alias EntryAlias -keypass EntryPassword
```

where *EntryAlias* is a name you want to assign for this key pair entry in the keystore and *EntryPassword* is a password required to guard against that key entry.

You will be prompted for the keystore password of your choice and some information to incorporate into the self-signed certificate. The newly generated public/private key pair and the self-signed certificate will be saved in the keystore file in the default location.

Refer to the JDK Security Tools documentation for details on the keystore.

2. Sign the JAR file with the private key.

Issue a command similar to the following from a console using the `jarsigner` utility:

```
console>jarsigner -storepass StorePass -keypass KeyPass  
CommonInterface.jar EntryAlias
```

where *StorePass* is the keystore password assigned while creating the public/private key pair entry in the previous step. *KeyPass* is the key pair entry password assigned in the previous step. *EntryAlias* is the name assigned to the key pair entry in the previous step.

This completes the applet signing process in preparation for the Common Interface to be run as an applet by the JVM plug-in of the platform browser. The signed JAR file contains a copy of the certificate from the keystore for the public key corresponding to the private key used to sign the JAR file.



### 3. Load the signed client into your browser.

Load the provided HTML page `CommonInterface.html` into your default browser. This page is an applet starter page that marks up the applet properties.

Accept the certificate to allow the browser JVM plug-in to execute the applet byte code. In a short time, the Common Interface should begin running, and you can interact with it within the boundary of the browser window.

---

## Using the Common Interface

Beginning in Acrobat SDK 7.0.5, PDF Library SDK 7.0.5, and Linux Reader version 7.0, you can interact with SnippetRunner by means of a common Java-based graphical user interface. (In previous versions, there were several interfaces: the ADM interface for Acrobat on Windows and Macintosh, and a command line interface for PDF Library on UNIX.)

Through the Common Interface, you can:

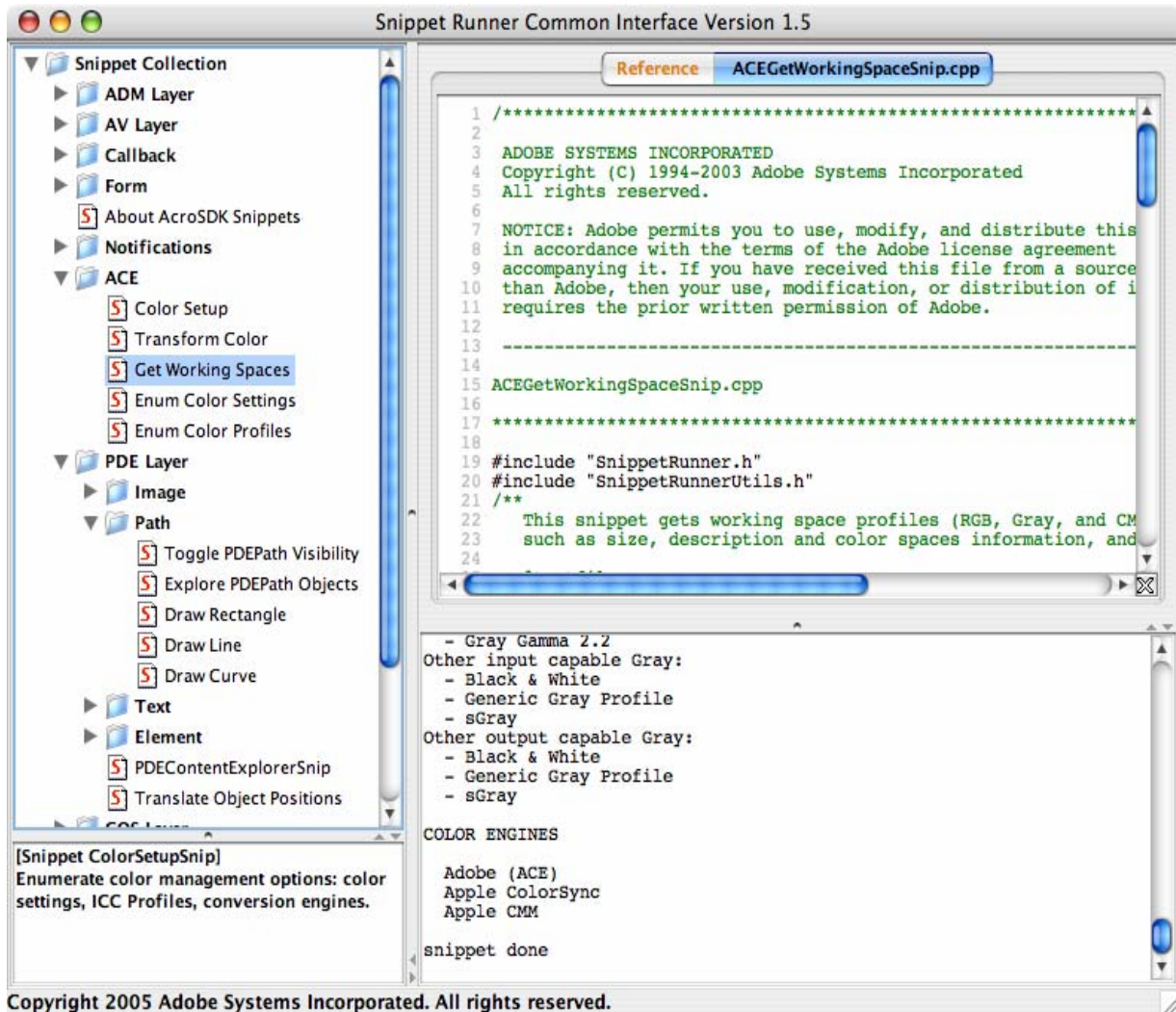
- See the collection of available code snippets sorted by categories
- Get snippet information
- Execute snippets
- Examine the output generated as a result of a snippet execution
- Browse snippet source code

You can use the keyboard or mouse to interact with the interface, as noted in the sections below.

**NOTE:** References to “right-click” usually mean Ctrl-click when running on Mac OS. However, on some Mac OS 10.3.x systems, this key combination may not work with SnippetRunner. If this occurs, using Command (Apple)-click will work.

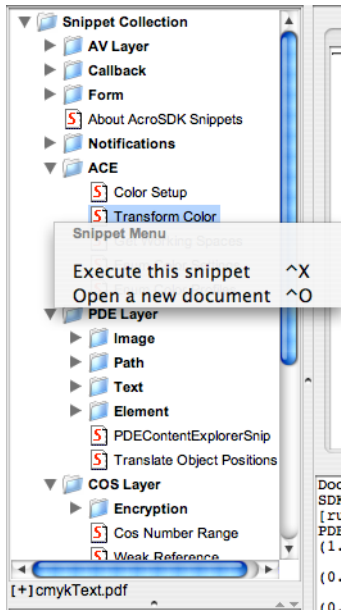
The Common Interface has four panes, as shown in the figure below:

- The Snippet Collection pane (upper left)
- The Snippet Description pane (lower left)
- The Source/Reference Browser pane (upper right)
- The Snippet Output pane (lower right)



You can resize, maximize, or minimize the main window as you would with any application. You can adjust the relative sizes of the individual panes by dragging the pane dividers. You can click on the arrow icons in the dividers to fully hide or expand a pane.

The Snippet Collection pane groups available snippets into a folder hierarchy for ease of access. These categories are defined in the snippets' registration macros (see "Understanding and Writing Snippets" on page 12).



You can navigate the hierarchy by means of mouse or keyboard.

- Use the Up/Down arrow keys to move up and down the list.
- Use the Left/Right arrow keys or click the triangles to expand or collapse a folder.

Whenever you select a snippet name, its description appears in the Snippet Description pane. If you double-click on a highlighted snippet name or press Enter or Return, its source code appears in the Source/Reference Browser Pane (see figure below). You can open multiple documents and switch between them.

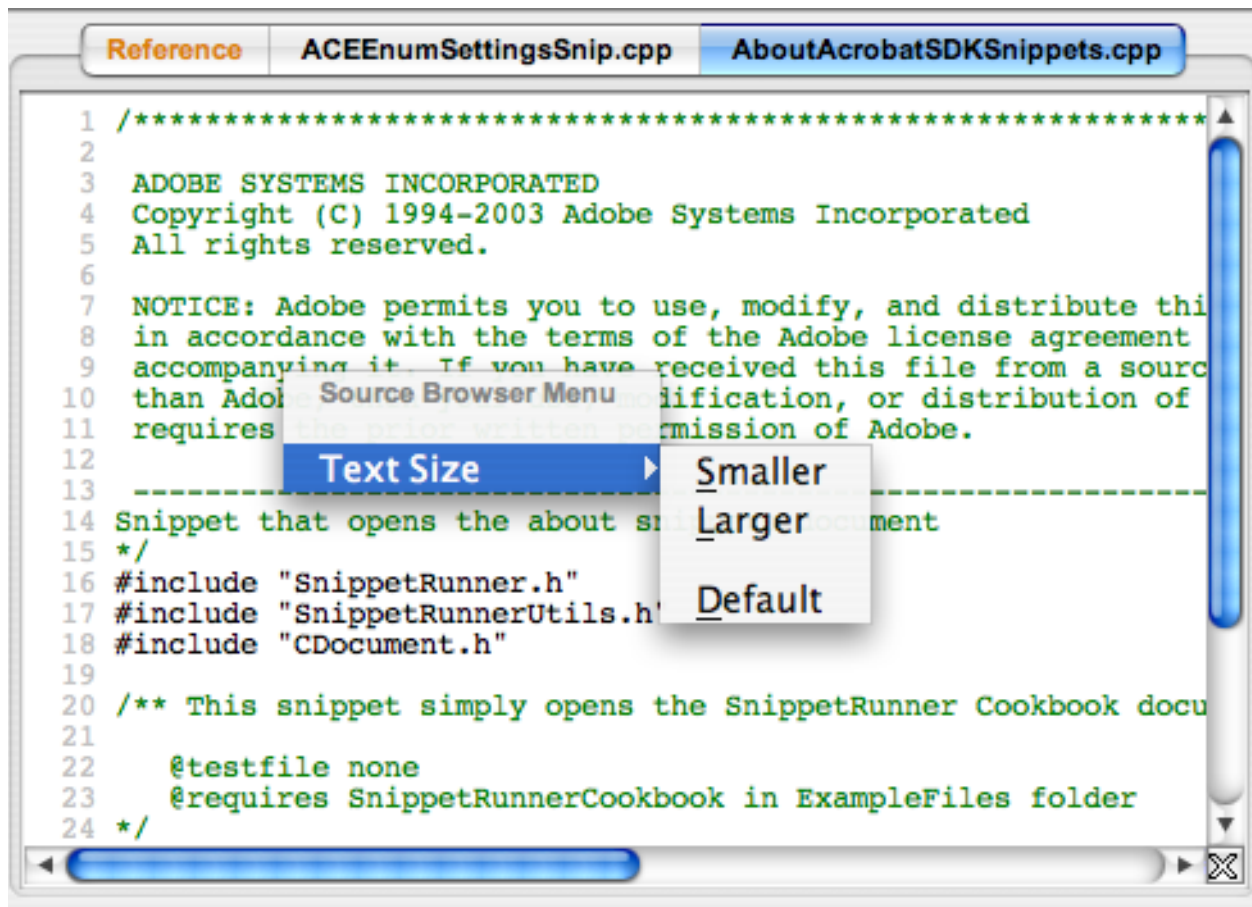
To execute a snippet, you can do one of the following:

- Right-click on the snippet name and select **Execute this snippet**.
- Select the snippet name and press Ctrl-X.

Output from executing the snippet is displayed in the Snippet Output pane. You can clear the output pane by right-clicking, then selecting **Clear output** from the context-sensitive menu.

In the PDF Library only, you can also right-click anywhere in the pane to select the command **Open a new document**, which brings up a file dialog to allow opening a document for use by a snippet.

At the bottom of the pane is a document status area that shows the file name of the current open document. A "+" in the brackets indicates that the document has been modified.



The Source/Reference Browser pane provides a tabbed interface to allow switching between the Reference view and the snippet source code views.

- The Source view displays the code for a selected snippet. You can switch between multiple snippets by clicking the tabs at the top of the window. Within this view, you can right-click to change the size of the text being displayed. You can close the specific source view by clicking the "X" in the lower-right corner.
- The Reference view displays SnippetRunner Cookbook documentation. You can navigate this document by means of mouse or keyboard. In addition, you can navigate between views of this document by right-clicking to access the Back and Forward commands in the context-sensitive menu (see figure below).

Reference ACEEnumSettingsSnip.cpp AboutAcrobatSDKSnippets.cpp

## SnippetRunners.

### Overview of SnippetRunner

The SnippetRunner plug-in is designed to allow developers to quickly prototype Acrobat API calls without the overhead of writing and verifying a complete plug-in. It provides an infrastructure and utility functions to support execution of Acrobat plug-in code snippets.

A plug-in code snippet is a small, self-contained portion of Acrobat plug-in code. In its simplest form, a snippet is a function demonstrating an Acrobat API call plus a SnippetRunner macro call to bind the snippet to the SnippetRunner environment. A snippet is required to include one function to act as its entry point, however, more complex snippets may contain additional functions.

The SnippetRunner's plug-in infrastructure supports execution of plug-in code snippets by providing a convenient environment in which you can test

Navigate Menu

Back

Forward

#### Known Issues

- The socket communication between the SnippetRunner server and client may be lost during sleep mode. To re-establish communication, restart the Common Interface.
- If you start the Acrobat process by loading a PDF document into the browser plug-in, the Common Interface socket communication will carry on with that process, which is most likely not expected.
- Currently, only Internet Explorer supports resizable content. All other browsers render the Common Interface at a fixed dimension.
- On Mac OS X, if you minimize the Common Interface window to the Dock and then bring it back into view, the Common Interface window might not be properly repainted. Resize the window to force the GUI refresh.
- On Windows, an Acrobat dialog triggered as a result of a command issued by the Common Interface may not come up to the top of all open windows, in which case the Common Interface may seem frozen. Simply bring up Acrobat to dismiss the dialog.

## Understanding and Writing Snippets

A code snippet must contain at least the following:

- A single main function that acts as its entry point.  
Snippets may contain additional functions as needed.
- A macro call that binds the snippet to the SnippetRunner environment.

An example of a single-function snippet that is included with the SnippetRunner project is `SimpleSnip.cpp`. It is shown below. The code for this snippet also includes comments (not shown here) that provide useful development hints.

```
#include "SnippetRunner.h"
#include "SnippetRunnerUtils.h"

void SimpleSnip()
{
    Console::displayString("This is a simple snippet.");
    Console::displayString("Simple snippet executed\n");
}

SNIPRUN_REGISTER(SimpleSnip, "Simple Framework", "SimpleSnip creates a
framework for a snippet.")
```

This snippet has a single function, `SimpleSnip`, which writes two messages to the output pane of the Common Interface (see [“Using the Common Interface” on page 7](#)) using the utility function `Console::displayString`. This function enables you to perform memory dumps and view strings (`char *`), `ASText` objects, and `Cos` objects. (See the source files `SnippetRunnerUtils.cpp` and `Console.cpp` for details.)

Because the `SimpleSnip` function requires no parameters, it uses the macro call `SNIPRUN_REGISTER` to bind to the SnippetRunner environment. (See [“Passing Parameters to Snippets” on page 13](#) for other possibilities.) This macro requires three parameters:

- The name of the function that is the entry point for the snippet (by convention, it is the same as the snippet’s file name).
- A string indicating where the snippet’s node is to appear in the Snippet Collection pane of the Common Interface UI.

If the snippet location is not at the root level of the hierarchy, the string specifies the path to the snippet, with folder names separated by colons. For example, the `GetFontInfoSnip` snippet would specify: `“PD Layer:Fonts:Get Font Info”`.

**NOTE:** A snippet’s node name is limited to 49 characters.

- A string of descriptive text to be presented in the Snippet Description pane.

`SimpleSnip` is a *synchronous* or “one-shot” snippet, meaning that it executes and then terminates. See [“Toggling and Asynchronous Snippets” on page 13](#) for other possibilities.

## Passing Parameters to Snippets

You can pass parameters to a snippet's main function. To enable this mechanism, use the `SNIPRUN_REGISTER_WITH_DIALOG` macro in your snippet to bind the snippet to `SnippetRunner`. This call takes an extra parameter (beyond the three required by `SNIPRUN_REGISTER`), which is a single string representing default parameter(s) separated by spaces.

When a snippet implemented with `SNIPRUN_REGISTER_WITH_DIALOG` is invoked, a dialog box with the snippet's descriptive text appears that includes a text edit box pre-populated with the default values set by the fourth parameter.

**NOTE:** If necessary, this dialog box can be suppressed using the utility calls `turnParamDialogOff (/On)` and `toggleParamDialog` (see `SnippetRunnerUtils.cpp`).

An example of the use of this macro and its resulting values is in the `TextChangeColour` snippet, whose macro call is written as follows:

```
SNIPRUN_REGISTER_WITH_DIALOG(TextChangeColourSnip, "PDE Layer:
Text:Change colour", "Shows how to change the colour of text
in a document", "0 0 65000")
```

When this snippet is invoked, a dialog box displays "Shows how to change colour of text in a document" and the parameter text edit box is pre-populated with the default values: "0 0 65000". For this example, the parameters are meant to represent RGB color values. You can edit the text in the dialog box to change the values of the parameters.

To access the parameters passed in through the dialog box, use the `ParamsManager` class. This class (see `ParamManager.cpp`) provides a set of methods that allow you to obtain the input parameters as integer, string, hex, and fixed data types. (To provide support for other data types, you must extend the `ParamsManager` class.)

For example, the `TextChangeColour` function is defined with a single parameter of type `ParamManager *`, to provide storage for the snippet's parameters:

```
void TextChangeColourSnip(ParamManager * thePM)
```

The following code in the `TextChangeColourSnip` function converts the input parameter string to three separate RGB values of type integer:

```
ASInt32 red, green, blue;

thePM->getNextParamAsInt (red);
thePM->getNextParamAsInt (green);
thePM->getNextParamAsInt (blue);
```

## Toggling and Asynchronous Snippets

**NOTE:** This section applies to Acrobat plug-in snippets only, not PDF Library SDK.

`SnippetRunner` provides utility methods for toggling behavior. For example, `FormCalculationsSnip` turns on and off the ability to perform form calculations. It uses the

**toggleSnippetCheck** method (see `SnippetRunnerUtils.cpp`) to turn the state ON if it was previously OFF, and vice versa.

Other snippets that toggle behavior include `AVPageViewToggleWireframeDrawingSnip`, and `AVAppShowAnnotProperties`.

Some snippets define and register callbacks in the same manner as plug-ins. (See the SDK documentation regarding **ASCallback** objects, **ASCallbackCreateProto** and **ASCallbackDestroy**). Specifically, to register a snippet for a notification, use **AVAppRegisterNotification** and provide a callback function with the appropriate arguments. To register your snippet for a specific event, such as `IdleProc`, `PageViewDrawing`, `PageViewClicks` or `PageViewAdjustCursor`, use the related **AVAppRegisterXXX** method. You can toggle a snippet to OFF by checking for its ON state and unregistering via the complementary **AVAppUnregisterXXX** method.

Such snippets can be *asynchronous* in the sense that they register a callback whose output (or other result) does not appear until a particular event occurs. Snippets that register for notifications include: `OptContNotificationTracerSnip`, `AVAppFrontDocChangeNotSnip`, `AVAppRegisterForPageViewDrawingSnip`, `PDDocDidDeletePagesNotSnip` and `IdleProcSnip`.

## Exception Handling

The `SnippetRunner` provides an exception handler that reports the name of the snippet that caused an exception. Synchronous snippets require no special considerations with regard to exception handling within the `SnippetRunner` environment.

However, if you write a snippet containing a callback that is called asynchronously, the callback function should include its own exception handlers to trap and handle various exceptions. When an exception occurs, your exception handler can perform any necessary cleanup, such as releasing memory. The core API provides the following macros for handling errors: **DURING\_HANDLER**, **END\_HANDLER** and **E\_RETURN**. If methods in your snippet code could return an error code or NULL if something goes wrong, you can call the **ASRaise** method, which generates an exception.

## Document Handling

`SnippetRunner` provides a C++ class, **CDocument**, that handles getting documents in the `SnippetRunner` environment. (See `CDocument.h` and `CDocument.cpp`.)

To use this class, a document must be open in the Acrobat viewer or PDF Library. You declare a **CDocument** object and then cast it to the type you need. For example:

```
CDocument document;  
AVDocGetFoo ( (AVDoc) document );
```

Supported cast types are:

- **AVDoc** - the front document (does not apply to PDF Library)
- **PDDoc** - the front **AVDoc** in Acrobat or the document that is open in PDF Library



- **CosDoc** - derived from the current **PDDoc**
- **AVPageView** - the current pageView in the front document (does not apply to PDF Library)
- **PDPage** - the page associated with the current page view in Acrobat (or the page that has been set in PDF Library). Defaults to the first page.

The destructor method of **CDocument** is called when the snippet returns. Therefore, you do not need to write code to release or destroy these objects.

