|  |  |
| --- | --- |
|  | **Stardock Corporation** |
|  | Last update date: 6/10/2009 |

|  |
| --- |
| **[DesktopX PLugin DOcumentation]** |
| Description of a DesktopX plugin and how to implement it. |

Table of Contents

[Overview 5](#_Toc232390418)

[General information 6](#_Toc232390419)

[Plugin Lifetime 7](#_Toc232390420)

[SDMessage 10](#_Toc232390421)

[SD\_INITIALIZE\_MODULE 10](#_Toc232390422)

[SD\_TERMINATE\_MODULE 10](#_Toc232390423)

[SD\_QUERY\_PLUGIN\_INFO 11](#_Toc232390424)

[SD\_QUERY\_CUSTOM\_STATES 11](#_Toc232390425)

[SD\_GET\_OBJECT\_SIZE 12](#_Toc232390426)

[SD\_CREATE\_PLUGIN 12](#_Toc232390427)

[SD\_DESTROY\_PLUGIN 12](#_Toc232390428)

[SD\_INITIALIZE\_PLUGIN 13](#_Toc232390429)

[SD\_TERMINATE\_PLUGIN 13](#_Toc232390430)

[SD\_DUPLICATE\_PLUGIN 13](#_Toc232390431)

[SD\_CONFIGURE 13](#_Toc232390432)

[SD\_SAVE\_DATA 14](#_Toc232390433)

[SD\_LOAD\_DATA 14](#_Toc232390434)

[SD\_STATE\_CHANGING 15](#_Toc232390435)

[SD\_STATE\_CHANGED 15](#_Toc232390436)

[SD\_DRAW 15](#_Toc232390437)

[SD\_COMMAND 16](#_Toc232390438)

[SD\_WINDOW\_MESSAGE 16](#_Toc232390439)

[SD\_MODE\_CHANGED 17](#_Toc232390440)

[Plugin flags 18](#_Toc232390441)

[SD\_FLAG\_DRAW 18](#_Toc232390442)

[SD\_FLAG\_SET\_SIZE 18](#_Toc232390443)

[SD\_FLAG\_CUSTOM\_ACTION 18](#_Toc232390444)

[SD\_FLAG\_CUSTOM\_STATES 18](#_Toc232390445)

[SD\_FLAG\_ONLY\_CUSTOM\_STATES 18](#_Toc232390446)

[SD\_FLAG\_FILTER\_STATE\_CHANGES 19](#_Toc232390447)

[SD\_FLAG\_SUBCLASS 19](#_Toc232390448)

[SD\_FLAG\_DRAW\_PPALPHA 19](#_Toc232390449)

[SD\_FLAG\_MORE\_INSTANCES 19](#_Toc232390450)

[Typical order of messages 20](#_Toc232390451)

[SDHostMessage 21](#_Toc232390452)

[SD\_GET\_OBJECT\_DIRECTORY 21](#_Toc232390453)

[SD\_GET\_PLUGINS\_DIRECTORY 21](#_Toc232390454)

[SD\_LOAD\_IMAGE 22](#_Toc232390455)

[SD\_REDRAW 22](#_Toc232390456)

[SD\_GET\_ABSOLUTE\_RECT 22](#_Toc232390457)

[SD\_GET\_RELATIVE\_RECT 23](#_Toc232390458)

[SD\_GET\_STATE 23](#_Toc232390459)

[SD\_SET\_STATE 23](#_Toc232390460)

[SD\_SET\_TOOLTIP\_TEXT 24](#_Toc232390461)

[SD\_REGISTER\_FILE 24](#_Toc232390462)

[SD\_CREATE\_OBJECT 24](#_Toc232390463)

[SD\_PACK\_ALPHA 25](#_Toc232390464)

[SD\_SET\_OBJECT\_POS 25](#_Toc232390465)

[SD\_ENUMERATE\_OBJECTS 25](#_Toc232390466)

[SD\_DELETE\_OBJECT 26](#_Toc232390467)

[SD\_OBJID\_FROM\_GUIID 26](#_Toc232390468)

[SD\_GUIID\_FROM\_OBJID 26](#_Toc232390469)

[SD\_SET\_OBJECT\_GUIID 27](#_Toc232390470)

[SD\_USER\_MODE 27](#_Toc232390471)

[SD\_MOVE 27](#_Toc232390472)

# Overview

SDPlugins are designed to allow you to extend the functionality of applications that support them such as DesktopX and ObjectBar. Examples of such plugins would be virtual desktops, MP3 players, analog clocks, email checkers, recycle bin checkers, printer status monitors, games, disk space monitors, CPU graphs, and anything else that needs to make calls to the system or the Internet and talk back to the object it is attached to or other objects running on the system. An SDPlugin will always attach itself to the object (or bar) that it is part of. And when exported for use by others, it will be included.

This guide is designed to document how SDPlugins are created. It is designed for software developers who are already familiar with a programming language and how to create DLLs.

# General information

Plugins must be compiled with **stdcall** calling convention.  
  
If you are compiling with MS Visual C++ you should also use “Multithread DLL” run-time library for Release version and “Debug Multithread DLL” run-time library for the Debug version, since the required modules are distributed with DX anyway and this way the plug-in will be a lot smaller.  
  
The constants mentioned in this document are declared in the [SDPlugin.h](http://code.google.com/p/threeoaks/source/browse/trunk/DesktopX/Plugins/SDK/SDPlugin.h) and [SDScriptedPlugin.h](http://code.google.com/p/threeoaks/source/browse/trunk/DesktopX/Plugins/SDK/SDScriptedPlugin.h) header files. You can find a few examples of DesktopX plugins in [this repository](http://code.google.com/p/threeoaks/source/browse/#svn/trunk/DesktopX/Plugins).

# Plugin Lifetime

This chapter discusses the differences between the plugin lifetime and the instance (of the plugin) lifetime.  
  
The plugin dll is loaded when an object needs the plugin. The [SD\_INITIALIZE\_MODULE](#_SD_INITIALIZE_MODULE) is sent through the SDMessage callback (see below).  
  
The plugin dll is unloaded when the last object using the plugin is deleted or the plugin removed, so no object needs the plugin. [SD\_TERMINATE\_MODULE](#_SD_TERMINATE_MODULE) is sent through the [SDMessage](#_SDMessage).  
  
When a plugin is selected to be used in an object an instance of the plugin is created and associated to the object. More instances can be added to an object (i.e. an email checking plugin can be added more times to handle more email accounts).  
  
When an instance of the plugin is created the plugin receive [SD\_CREATE\_PLUGIN](#_SD_CREATE_PLUGIN). Note that at this point the plugin should not start its action (i.e. a clock plugin should not start displaying the hour), i.e. the DesktopX object is not yet created. This message is sent for the plugin to initialize instance data, allocate variables and such.  
  
The plugin instance actually starts working when it received the [SD\_INITIALIZE\_PLUGIN](#_SD_INITIALIZE_PLUGIN) message. In fact now it receives the object HWND and the RECT size of the object. It means that the object has been created and the plugin should start “running”.  
  
After the instance is created ([SD\_CREATE\_PLUGIN](#_SD_CREATE_PLUGIN)) but before it is running ([SD\_INITIALIZE\_PLUGIN](#_SD_INITIALIZE_PLUGIN)), the plugin can be configured. This can happen in three ways:

* Directly by the user: the user just clicked “Configure” in the plugin section of the object properties. At this point [SD\_CONFIGURE](#_SD_CONFIGURE) is sent to the plugin and it would typically open a configuration dialog to configure the instance settings.
* When an object is loaded from the disk (i.e. the user loads a theme or package with objects using your plugin). The instance configuration data is typically loaded from configuration files that the plugin previously saved in the theme directory and registered to the host (so that they are saved in themes or packages). The plugin knows that it should retrieve the saved instance configuration data when it receives the [SD\_LOAD\_DATA](#_SD_LOAD_DATA) message.
* By the host through [SD\_DUPLICATE\_PLUGIN](#_SD_DUPLICATE_PLUGIN) message. This is probably the only hard part of the whole specs. Basically [SD\_DUPLICATE\_PLUGIN](#_SD_DUPLICATE_PLUGIN) is used overall by DesktopX to support special operations: when an object is duplicated and when an object is configured. On cloning an object, a new object is created and so, new plugin instances are created. These new instances need to be configured the same as the original object/instances. In fact [SD\_DUPLICATE\_PLUGIN](#_SD_DUPLICATE_PLUGIN) contains references to the original plugin instance to copy (in the sense of “configuring” it).

Why is [SD\_DUPLICATE\_PLUGIN](#_SD_DUPLICATE_PLUGIN) needed on configuring an object? When you open the object properties panel, you can still use all existing objects, including the object you are configuring. You can change its graphics, settings, add, remove, configure plugins and the configured object is still working the same way. All changes are applied when you actually click “Apply” or “OK”. If you click “Cancel” the changes are ignored.

To support this mechanism DesktopX performs these tasks:

* + When the user opens the object properties panel, a new “dummy” plugin instance is created. [SD\_DUPLICATE\_PLUGIN](#_SD_DUPLICATE_PLUGIN) is sent to configure this new instance in the same way as the original instance.
  + This “dummy” instance is used to be configured, but will never actually receive [SD\_INITIALIZE\_PLUGIN](#_SD_INITIALIZE_PLUGIN).
  + When the user clicks “Apply”, the original object is destroyed with all his plugin instances. A new object is created with new settings (graphics, options, and new added plugins, removed plugins or differently configured plugin instances). New plugin instances are now created for the object. These are the final “real” instances and they are configured with [SD\_DUPLICATE\_PLUGIN](#_SD_DUPLICATE_PLUGIN): the new plugin instance need to be configured the same as the “dummy” instances used in the properties panel.
  + The host destroys the “dummy” instances, while the new instances receive [SD\_INITIALIZE\_PLUGIN](#_SD_INITIALIZE_PLUGIN) and start working.

This could sounds difficult, but it is just to explain how the host works with plugins, you can simply ignore all of the above explanations and just follow and respect the plugin messages. Simply keep in mind that [SD\_CREATE\_PLUGIN](#_SD_CREATE_PLUGIN) initializes the plugin configuration data (you would set this to default settings), [SD\_CONFIGURE](#_SD_CONFIGURE) configures this data through the user’s input, [SD\_DUPLICATE\_PLUGIN](#_SD_DUPLICATE_PLUGIN) configures a plugin instance the same way as another plugin instance, [SD\_LOAD\_DATA](#_SD_LOAD_DATA) configures the plugin instance from disk saved data. [SD\_INITIALIZE\_PLUGIN](#_SD_INITIALIZE_PLUGIN) finally “starts” the plugin (i.e. a timer starts running, the email check plugin starts checking for emails, etc.).  
  
When the plugin instance is removed by the object, or the object is deleted, [SD\_TERMINATE\_PLUGIN](#_SD_TERMINATE_PLUGIN) is sent for each instance.  
  
Note that some configuration data is also needed by the host. For example, if the plugin is a drawing plugin (i.e. it draws the object graphics – the analog clock being one example), the host needs to know it to correctly allocate the graphic buffers to be used by the plugin. So, the host needs to know if the plugin will subclass the object. Plugins pass this information to the host in two cases:

* When the plugin instance is created ([SD\_CREATE\_PLUGIN](#_SD_CREATE_PLUGIN)): the plugin should typically set default flags and/or flags that won’t change.
* When the plugin is configured ([SD\_CONFIGURE](#_SD_CONFIGURE)): the plugin can change the current flags (saved in [SD\_CREATE\_PLUGIN](#_SD_CREATE_PLUGIN)).

This way plugin options are “per instance” rather than “per plugin”. So, an instance could be set as drawing plugin, another instance of the same plugin could behave differently (same for the subclass option and the other ones).  
  
As mentioned earlier, plugins can store data in external files (configuration files, graphics, etc.). These files should be stored in the directory returned by [SD\_GET\_OBJECT\_DIRECTORY](#_SD_GET_OBJECT_DIRECTORY).

## SDMessage

*BOOL SDMessage(DWORD objID, DWORD \*pluginIndex, UINT messageID, DWORD param1, DWORD param2)*

|  |  |
| --- | --- |
| Parameter | Description |
| objID | Unique object identifier. It must be passed to the host when using callbacks |
| pluginIndex | Object identifier used by the plugin. It is modified by the plugin itself and is passed by the host on all messages. The plugin can use it to directly reference the plugin local data and settings |
| param1 | message-specific |
| param2 | message-specific |

SDMessage is used by the host to send callback messages to the plugin.  
  
Return *TRUE* if the message is supported and is processed.  
Return *FALSE* if the message is unsupported or not processed.

### SD\_INITIALIZE\_MODULE

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | BOOL (\_\_stdcall \*)(UINT, DWORD, DWORD) | SDHostMessage function pointer |
| param2 | HINSTANCE | dll instance |

This message is sent once when the plugin module is loaded by the host (i.e. should be called when a plugin is going to use this module and no other plugins already exist that use this module.  
  
The *param1* contains a pointer to the SDHostMessage function the plugin will use to communicate with the host.

### SD\_TERMINATE\_MODULE

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | N/A |
| param2 | DWORD | N/A |

This message is sent once when the plugin module is unloaded by the host (i.e. when the last plugin that uses this module is destroyed).

### SD\_QUERY\_PLUGIN\_INFO

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | SD\_PLUGIN\_INFO\* | Pointer to SD\_PLUGIN\_INFO structure |
| param2 | DWORD | N/A |

This message is sent when the host needs to obtain information about the plugin. The info is returned in the structure which is passed into *param1*. This structure is allocated by the host.  
  
Returns *TRUE* if the plugin is host compatible. Returns *FALSE* if the plugin isn't compatible with the host.  
  
Note: this message can be received before [SD\_INITIALIZE\_MODULE](#_SD_INITIALIZE_MODULE), so if the plugin absolutely needs to initialize before even being queried for info, it should use *DllMain* to initialize itself and not [SD\_INITIALIZE\_MODULE](#_SD_INITIALIZE_MODULE).

### SD\_QUERY\_CUSTOM\_STATES

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | char\* | Name of the custom state |
| param2 | int | Index of the custom state |

This message is sent by the host to enumerate the custom states supported by the plugin.

The name of the custom state should be copied into *param1*. Return TRUE to continue the enumeration, *FALSE* to cancel or stop the enumeration (*param1* will be ignored in this case).  
  
*Example:  
  
if(iCustomStateIndex == 0)  
 strcpy(szCustomState, “No Mail, Default”);  
else if(iCustomStateIndex == 1)  
 strcpy(szCustomState, “No Mail, Mouseover”);  
else if(iCustomStateIndex == 2)  
 strcpy(szCustomState, “You’ve Got Mail, Default”);  
 else if(iCustomStateIndex == 3)  
strcpy(szCustomState, “You’ve Got Mail, Mouseover”);  
else  
 return FALSE;  
return TRUE;*

### SD\_GET\_OBJECT\_SIZE

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | SIZE\* | Pointer to a size structure |
| param2 | DWORD | N/A |

This message is sent if the plugin is a drawing plugin. The plugin has already been associated to the object and it is already configured. The host must now know the desired object size.

### 

### SD\_CREATE\_PLUGIN

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD\* | Plugin flags |
| param2 | DWORD | N/A |

This message is sent when the plugin instance is created: the plugin can initialize instance data, allocate variables but should not begin his action, as the DesktopX object has not been created yet.

*pluginIndex* should be initialized here, allocating memory and storing pointers to that memory as needed.

### SD\_DESTROY\_PLUGIN

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | N/A |
| param2 | DWORD | N/A |

This message is sent when the plugin instance is destroyed.

*pluginIndex* should be de-initialized here; any memory associated with it should be freed, along with any other local data that was allocated.

### SD\_INITIALIZE\_PLUGIN

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | SD\_PLUGIN\_INIT\* | N/A |
| param2 | DWORD | N/A |

This message is sent when the plugin is initialized. The plugin should initialize plugin local data, timers, etc.

### SD\_TERMINATE\_PLUGIN

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | N/A |
| param2 | DWORD | N/A |

This message is sent when the plugin is being uninitialized. The plugin must destroy timers, etc.

### SD\_DUPLICATE\_PLUGIN

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | ObjectID of the original plugin |
| param2 | DWORD\* | pluginIndex of the original plugin |

This message is sent when the plugin is duplicated. The plugin should copy its configuration data to the new instance.

*Example:*

*// Copies over the memory from the old plugin’s configuration data to the new plugin*

*// Note that further processing might be needed if more pointers/handles are located*

*// within the memory pointed to by pluginIndex*

*CopyMemory((\*pluginIndex), (\*pluginIndex\_OriginalPlugin), sizeof(MYSTRUCT));*

*(\*pluginIndex) = (\*pluginIndex\_OriginalPlugin);*

### SD\_CONFIGURE

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD\* | Plugin flags |
| param2 | HWND | Handle of the parent window |

This message is sent when the plugin needs to be configured. The plugin will typically show a configuration window to modify its settings.

The plugin flags defined in [SD\_CREATE\_PLUGIN](#_SD_CREATE_PLUGIN_1) can also be modified to match this instance configuration.

### SD\_SAVE\_DATA

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | char\* | Instance ID of the plugin |
| param2 | BOOL | TRUE if the user is exporting, FALSE otherwise |

This message is sent when the plugin should save the configuration data, i.e. before unloading or before exporting.

The plugin will need to send [SD\_REGISTER\_FILE](#_SD_REGISTER_FILE) messages to the host for each file that needs to be associated with the object.  
  
The plugin should write in *param1* an instance identifier that the plugin will use at [SD\_LOAD\_DATA](#_SD_LOAD_DATA_1) time to identify the different instances if the plugin is associated to more than one object.  
  
If the user is exporting, the plugin could choose to save configuration data differently, i.e. an email notification plugin will not save the email login/password information since the objects could be distributed around the web.

### SD\_LOAD\_DATA

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | char\* | Instance ID of the plugin |
| param2 | DWORD\* | Plugin flags |

This message is sent when the plugin should load the configuration data for the object.

In param1 the host passes the identifier of the plugin object instance that the plugin itself wrote in [SD\_SAVE\_DATA](#_SD_SAVE_DATA). This is needed to identify plugin instances associated to different objects.

### SD\_STATE\_CHANGING

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | char\* | the state the plugin is changing to |
| param2 | DWORD | N/A |

This message is sent when the object state is about to change.

You can either return FALSE to allow the host to change the object current state to the new state, or return TRUE and prevent the host from changing the object state.  
  
If you decide to prevent state change and want to force the state to different one, you can copy the new desired state string identifier in param1. If an empty string is set, the plugin will simply cancel the state changing request.  
  
Some standard state identifiers are defined in SDPlugin.h.

### SD\_STATE\_CHANGED

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | char\* | the state the plugin has changed to |
| param2 | DWORD | N/A |

This message is sent when the object state has changed.

### SD\_DRAW

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | SD\_DRAW\_INFO \* | Pointer to a SD\_DRAW\_INFO structure |
| param2 | DWORD | N/A |

This message is sent when the plugin needs to redraw itself. The plugin must draw into the passed HDC.

Note that the plugin shouldn’t consider what is already in the HDC. It should draw a complete frame, eventually using a pink region (RGB = 255, 0, 255) where it needs transparency. After the plugin returns, the host will actually draw in the background, but this is not plugin task.  
  
If the host is DesktopX, the HBITMAP will always be a 32 bit memory DIB. In this case the plugin can directly modify the color. The bitmap so drawn is then passed to the DesktopX renderer, that is, the plugin will never draw to the HDC directly.  
  
If the plugin draws in opaque mode (not using per pixel information), it can use pink regions (RGB = 255, 0, 255) to indicate transparent regions.

### SD\_COMMAND

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | N/A |
| param2 | DWORD | N/A |

This message is sent when the plugin needs to perform a custom action.

The plugin will have to be registered to perform a custom action with the [SD\_FLAG\_CUSTOM\_ACTION](#_SD_FLAG_CUSTOM_ACTION) flag.

### SD\_WINDOW\_MESSAGE

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | MSG\* | Pointer to a MSG structure |
| param2 | DWORD\* | Return value |

This message is sent from the host if the plugin has registered itself to hook window messages.  
  
Note that in the current implementation the time and pt members of the MSG structure are not used.  
  
At the time this message is sent, the window message is not yet processed by the host. The plugin can simply retrieve the info or modify the message parameters.  
  
Return TRUE if the message must be not processed by the host. The plugin must write in param2 the desired value to return to the window procedure.  
Return FALSE if the message must be processed by the host.

### SD\_MODE\_CHANGED

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | BOOL | TRUE if the host is in User Mode, FALSE if in Edit/Design mode |
| param2 | DWORD | N/A |

This message is sent to all plugins when the current mode is changed.

# Plugin flags

The flags are used to configure the plugin instance. The default flag is set on [SD\_CREATE\_PLUGIN](#_SD_CREATE_PLUGIN_1) and can be changed on [SD\_CONFIGURE](#_SD_CONFIGURE_1) and [SD\_LOAD\_DATA](#_SD_LOAD_DATA_1).

### **SD\_FLAG\_DRAW**

The plugin will draw directly the object appearance. If a drawing plugin is selected for an object, the user won't be able to directly setup the messages/states in the GUI. The appearance will be only configurable through the plugin configuration dialog.

### SD\_FLAG\_SET\_SIZE

This flag can be used by drawing plugins, if the plugin needs to determine the initial object size, instead of using the one specified by the user in the object size settings (that will be disabled). In this case the plugin will receive [SD\_GET\_OBJECT\_SIZE](#_SD_GET_OBJECT_SIZE) message when the host needs to know those values to create or modify the window or do other internal adjustments.

### SD\_FLAG\_CUSTOM\_ACTION

The plugin will perform a custom action (i.e. control Winamp). It is a command plugin. If a command plugin is selected for an object, the command class will switch automatically to "Custom class" and the "Configure" button will open the plugin configuration panel. See [SD\_COMMAND](#_SD_COMMAND) for more details.

### SD\_FLAG\_CUSTOM\_STATES

The plugin has custom states that it wants the host to observe

### SD\_FLAG\_ONLY\_CUSTOM\_STATES

Items/objects using this plugin should only show the custom states when the users is editing the item's appearance.

### SD\_FLAG\_FILTER\_STATE\_CHANGES

The plugin receives [SD\_STATE\_CHANGING](#_SD_STATE_CHANGING) messages that inform it about state changes. The plugin can modify and/or remove them as they get sent.

### SD\_FLAG\_SUBCLASS

The plugin should set this flag to indicate if it wants to get and set messages from the window procedure sent from the host through SD\_WINDOW\_MESSAGE. Note that the plugin MUST NOT subclass the object window. This hooking service should be used instead.

### SD\_FLAG\_DRAW\_PPALPHA

The plugin informs the host that it will use the 4th byte of the 32bit HBITMAP to supply per-pixel alpha blending information.

### SD\_FLAG\_DRAW\_DOUBLE\_HEIGHT

The host will provide a double-height 32 bit HBITMAP for the plugin to draw alpha blending values in the bottom half, instead of in the fourth channel. This is deprecated and not supported anymore in recent versions of DesktopX.

### SD\_FLAG\_MORE\_INSTANCES

By default the host will avoid that more instances of the same plugin are added to an object. This flag will make it possible to add more than one instance.

**SD\_FLAG\_NO\_BUILDER\_CONFIG**

The plugin will have no configuration UI available in builder mode.

**SD\_FLAG\_NO\_USER\_CONFIG**

The plugin will have no configuration UI available in builder mode.

# Typical order of messages

|  |  |
| --- | --- |
| Message | What it does |
| SD\_QUERY\_PLUGIN\_INFO | Can be called only for querying purpose, in this case the other messages won’t be called. |
| SD\_INITIALIZE\_MODULE | Only called if this is the first plugin from this DLL being loaded. |
| SD\_CREATE\_PLUGIN | Setup pluginIndex, allocate memory |
| SD\_LOAD\_DATA |  |
| SD\_CONFIGURE | These are the two ways a plugin instance can be configured. The plugin can modify the plugin flags here. |
| SD\_QUERY\_CUSTOM\_STATES | Custom states actually depend on the plugin configuration, so these should be set when the plugin is first time created and when it is configured. |
| SD\_INITALIZE\_PLUGIN |  |
| SD\_QUERY\_CUSTOM\_STATES | Custom states actually depend on the plugin configuration, so these should be set when the plugin is first time created and when it is configured. |
| SD\_SAVE\_DATA | Save plugin data to object directory or plugin directory. Can be called at any time between SD\_INITIALIZE and SD\_TERMINATE. |
| SD\_TERMINATE\_PLUGIN | Un-initialize plugin, stopping any times it has, etc.. |
| SD\_DESTROY\_PLUGIN | Destroy pluginIndex, deallocate memory. |
| SD\_DEINIALIZE\_MODULE | Only called if this is the last plugin from this DLL being unloaded. |

In order to support Apply/Cancel/Ok in the object configuration panel the plugin is requested to duplicate its plugin data through [SD\_DUPLICATE\_PLUGIN](#_SD_DUPLICATE_PLUGIN).

## SDHostMessage

*DWORD SDHostMessage (UINT messageID, DWORD param1, DWORD param2)*

|  |  |
| --- | --- |
| Parameter | Description |
| *messageID* | Unique message identifier |
| param1 | message-specific |
| param2 | message-specific |

SDHostMessage is used by the plugin to communicate with the host or request certain tasks (i.e. move an object, get the theme directory, etc.).  
  
The pointer to this function is passed to the plugin in the [SD\_INITIALIZE\_MODULE](#_SD_INITIALIZE_MODULE) message.

### 

### SD\_GET\_OBJECT\_DIRECTORY

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | char\* | char buffer to receive the object directory |
| param2 | DWORD | Object ID |

The object directory is the directory that will contain the plugin custom files if needed (files that will be registered through [SD\_REGISTER\_FILE](#_SD_REGISTER_FILE)).

This folder should be part of the theme, and should be the folder where the object/theme is being saved to. This is designed to save object/theme-specific “settings” which change from object to object, theme to theme, like the color of a clock.

### SD\_GET\_PLUGINS\_DIRECTORY

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | char\* | char buffer to receive the plugins directory |
| param2 | DWORD | Object ID |

The host will fill the char buffer with the path of the common plugins folder for DX or for OB… like DesktopX\SDPlugins\ or ObjectBar\Plugins\. The plugin could then create subfolders for itself as needed, so it can store data common to everything that uses this plugin.  
  
Again this is here so the plugin knows where to save general “data”, which should be common no matter what theme/object uses it, such as for the command line plugin’s history, etc.

Note: As of Windows Vista, writing to the plugin folder is no longer possible in most cases because of UAC. This is still useful to read external data files that were installed with the plugin, if any.

### SD\_LOAD\_IMAGE

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | SD\_IMAGE\_INFO\* | SD\_IMAGE\_INFO structure |
| param2 | DWORD | N/A |

Loads an .ico, .bmp, .jpg or .png (all formats supported) file to a 32 bit bitmap.

You can pass a custom width to extract a specific icon size if the file is an icon.

### SD\_REDRAW

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | Object ID |
| param2 | DWORD | N/A |

Sends a request to update/redraw the object (i.e. the clock plugin would call this each second).  
  
The plugin will receive back a [SD\_DRAW](#_SD_DRAW) message when the host is ready to have the object redrawn.  
  
Depending of the implementation of the host, the plugin can receive [SD\_DRAW](#_SD_DRAW) back immediately or not.

### SD\_GET\_ABSOLUTE\_RECT

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | Object ID |
| param2 | RECT\* | pointer to a structure to store the rectangular area used |

The plugin should pass a pointer to a RECT structure in param2, which the host will fill with the rectangular area of the screen occupied by the container HWND, the object/bar that owns the plugin whose ID is passed in param1.

### SD\_GET\_RELATIVE\_RECT

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | Object ID |
| param2 | RECT \* | pointer to a structure to store the rectangular area used |

The plugin should pass a pointer to a RECT structure in param2, which the host will fill with the rectangular area occupied by the object in the HWND (the container). This cannot be the same rectangle passed on [SD\_DRAW](#_SD_DRAW), because that rectangle refers to the area occupied in the passed HDC.  
  
This allows the plugin to process mouse handling and such if it subclasses, and could also be used for some other purposes as well.

### SD\_GET\_STATE

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | Object ID |
| param2 | char\* | the current object state |

The host copies in param2 the current state identifier.

### SD\_SET\_STATE

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | Object ID |
| param2 | char\* | the new object state to set |

The plugin should pass a char buffer to the host to set the current state to that custom state if it exists.

### SD\_SET\_TOOLTIP\_TEXT

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | Object ID |
| param2 | char\* | the new tooltip text |

The plugin should pass a char buffer to set the current tooltip text of the object.

### SD\_REGISTER\_FILE

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | char\* | path to the file to register |
| param2 | DWORD | N/A |

The plugin should send this message to the host while processing the [SD\_SAVE\_DATA](#_SD_INITIALIZE_PLUGIN) message. This message will need to be sent once for every file that needs to be registered. The plugin will need to register all files that will need to be saved on every [SD\_SAVE\_DATA](#_SD_INITIALIZE_PLUGIN) message.

### SD\_CREATE\_OBJECT

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | char\* | Object definition |
| param2 | DWORD | IO version |

The plugin calls this to create an object. It must make sure all required graphic and sound files are present in the theme directory.  
  
The Object definition should contains the multiline (\r separated) object string from “OBJECT” to “!OBJECT”.  
  
The IO version is the version of the IO syntax used in the object definition. You should check what version is used by the latest DX version and use the same number and latest syntax. You can find the version number in the first line of theme.ini.

### SD\_PACK\_ALPHA

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | HBITMAP\* | pointer to the HBITMAP to convert |
| param2 | DWORD | N/A |

The plugins call this to converts a double height 32 bit bitmap to a single height 32 bit bitmap with packed alpha.

### SD\_SET\_OBJECT\_POS

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | Object ID |
| param2 | SD\_SOP\_INFO\* | the object position |

The plugin should pass a SD\_SOP\_INFO structure containing the new object position along with a set of flag to determine the object state after the move.

You can use one or more of this flags:

* SD\_SOP\_MOVE: x and y are used
* SD\_SOP\_SIZE: width and height are used (drawing plugins only)
* SD\_SOP\_SHOW: shows the object
* SD\_SOP\_HIDE: hides the object
* SD\_SOP\_ONTOP: sets the object on top of its zorder class, i.e. if it is a desktop level object, it will be placed on top of all other desktop level objects

### SD\_ENUMERATE\_OBJECTS

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | callback function pointer |
| param2 | DWORD | N/A |

The plugin should pass a pointer to the callback procedure to enumerate all the loaded objects.  
  
The callback function should have the following signature:

BOOL (CALLBACK \*EnumObjectsProc) (DWORD ObjID, DWORD lParam);

To continue enumeration, the callback function must return TRUE; to stop enumeration, it must return FALSE.

### SD\_DELETE\_OBJECT

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | Object ID |
| param2 | BOOL | TRUE to delete children, FALSE otherwise |

The plugin should call this message to delete an object.

Returns TRUE if no errors occurred, returns FALSE if: the objID was invalid or the object was being configured.

### SD\_OBJID\_FROM\_GUIID

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | char[100] | the Object GUIID |
| param2 | DWORD | Object ID |

Get an object ID from an object GUIID (defined by the user in the GUI).

### SD\_GUIID\_FROM\_OBJID

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | Object ID |
| param2 | char[100] | the Object GUIID (out) |

Get an object GUIID from its ID. The host copies in param2 the object ID defined by the user in the GUI.

### SD\_SET\_OBJECT\_GUIID

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | Object ID |
| param2 | char[100] | the Object GUIID |

Set an object GUIID. The plugin should pass the new object GUIID in param2

### SD\_USER\_MODE

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | N/A |
| param2 | DWORD | N/A |

Returns TRUE if DesktopX is in User mode, FALSE if it is in Design/Edit mode.

### SD\_MOVE

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| param1 | DWORD | ?? |
| param2 | DWORD | ?? |

 ??

Scriptable plugins

SD\_REGISTER\_SCRIPTABLE\_PLUGIN

SD\_SCRIPTABLE\_PLUGIN\_EVENT

There are a few things to note when writing scriptable plugins:

* There is no configuration management. It is left to the actual script.
* It uses the special SD\_SCRIPTABLE\_PLUGIN\_EVENT message to fire script events. This is supposed to solve threading issues as compared to actual native event mechanisms. With this regard, DesktopX is serving as an event proxy.
* There is a custom implementation of DllRegisterServer all scriptable plugins should comply with. It is basically registering the plugin information using HKEY\_CURRENT\_USER rather than HKEY\_LOCAL\_MACHINE. This allows auto-setup when running a gadget under limited user privileges.