

# Technical Note TN1170

## The Printing Plug-ins Manager

### CONTENTS

[About This Technote](#)[Information for Plug-in Creators](#)[Printing Plug-ins Manager APIs](#)[Utility Routines](#)[Errors](#)[References](#)[Change History](#)[Downloadables](#)

New to PrintingLib 8.6 is functionality built into the SettingsLib library contained in the PrintingLib file. This functionality allows straightforward use of plug-in modules contained in either the PrintingLib file or in a special folder called "Printing Plug-ins" in the Extensions folder in the user's System folder. Plug-in modules are shared libraries whose behavior is unspecified.

The "Printing Plug-ins Manager" is intended to make it straightforward for software clients to create, load, and use shared libraries which can be treated as printing plug-ins. The Download Manager has been implemented using the Printing Plug-ins Manager to manage its plug-ins. Future printing software will use the Printing Plug-ins Manager to manage other plug-ins.

This Technote is directed at application developers who wish to write plug-in modules.

Updated: [May 24 1999]

---

## About This Technote

This document is divided into two main sections: [Information for Plug-in Creators](#) and [Plug-ins Manager APIs](#).

The section [Information for Plug-in Creators](#) describes the requirements the Printing Plug-ins Manager imposes on the files it manages. These requirements are minimal and are probably similar to requirements a client would otherwise impose. This section applies to developers of Download Manager plug-in low level converters, Desktop Printer Utility plug-ins, and Custom Hoses since these plug-ins are managed by the Printing Plug-ins Manager.

The section [Printing Plug-ins Manager APIs](#) describes the APIs used by clients of the Printing Plug-ins Manager. This section consists of a "simple" call to obtain a list of plug-ins of given type, a more complex version of this call for

specialized clients, and a set of utility routines. This section applies only to developers who wish to put plug-ins into the Printing Plug-ins Folder and use the [Printing Plug-ins Manager APIs](#) to manage them. For example, an installer program for a plug-in module might use the `psGetPrintingPluginsFolder` routine to install printing plug-ins into the proper folder for use with the Download Manager.

Please see [Technote 1169: The Download Manager](#) for information about one client of the Printing Plug-ins Manager.

[Back to top](#)

## Information for Plug-in Creators

Plug-ins managed by the Printing Plug-ins Manager are required to have a resource of type 'PLGN' with ID -8192 that contains information about the library. If they do not, they cannot be used and are ignored by the Printing Plug-ins Manager. They are also required to have a standard 'cfrg' resource describing the code fragments in the data fork of the file.

The 'PLGN' resource contains information about how many plug-in shared libraries are contained in this file and for each shared library the type of plug-in that it is, the subtype that library handles and the library name. A subtype of '????' means that particular plug-in can handle multiple subtypes (as is the case for the Download Manager Plug-ins).

The 'PLGN' resource is defined as follows (using Rez syntax):

```
type 'PLGN' {
  integer = $$Countof(PluginLibInfo);
  array PluginLibInfo {
    literal longint; /* Type */
    literal longint; /* subtype */
    pstring; /* library name */
    align word;
  };
};
```

A ResEdit template resource ('TMPL') for the 'PLGN' resource is contained within PrintingLib versions 8.6 and later.

The PluginLibInfo structure in C syntax is:

```
typedef OSType SettingsDataType;
typedef OSType SettingsDataSubType;

short num; // the number of shared libraries this 'PLGN' describes
PluginLibInfo libInfo[num];

typedef struct PluginLibInfo{
  SettingsDataType type;
  SettingsDataSubType subtype;
  unsigned char libraryName[ ]; // pascal string
  // word aligned
}PluginLibInfo;
```

`type` is the type of plug-in that is described by this `PluginLibInfo`.

`subtype` is the subtype of data that can be handled by the plug-in described by this `PluginLibInfo`.

`libraryName` is the library name of the code fragment in the plug-in file described by this `PluginLibInfo`.

**Note:**

A single file can contain multiple plug-in libraries.

**Note:**

The subtype `'zsys'` is reserved for use by the Printing Plug-ins Manager and cannot be used by a client.

A file containing plug-ins for use with the Printing Plug-ins Manager is not required to have any specific Finder Type or Creator. However, the Finder Type `'bird'` is a type that is auto-routed by the Finder into the Printing Plug-ins Folder when Mac OS 8.5 and PrintingLib 8.6.1 (or later) are used. For this reason, it is recommended that developers of Printing Plug-ins use `'bird'` as the Finder Type for the file. Note that the Finder Type has no connection to the `SettingsDataType` referred to by the `PluginLibInfo` contained in the `'PLGN'` resource.

[Back to top](#)

## Printing Plug-ins Manager APIs

### Definitions

`ALibraryLocator` is a structure which can be used by a client to open a particular shared library using `GetDiskFragment`. It contains information about the `FSSpec`, the library name, and for the currently executing architecture, the offset of the shared library in the file and the library's length.

```
typedef struct LibraryLocator{
    FSSpec  librarySpec;
    Str63   libraryName;
    SInt32  libraryOffset; // for the currently executing
                        // architecture
    SInt32  libraryLength; // for the currently executing
                        // architecture
}LibraryLocator;
```

For many operations, the Printing Plug-ins Manager maintains a cache of information about the installed Printing Plug-ins. The cache is updated as new plug-ins are installed or existing ones are removed or updated. This cache allows the Printing Plug-ins Manager to respond quickly in its search for a plug-in of a requested type and subtype.

### Simple Case

```
OSStatus psGetSubtypeLocators(SettingsDataType type,
    SettingsDataSubType subtype, long *numLocatorsP,
    LocatorListH *locatorListHP);
```

A client calls this routine to get a handle containing a list of `LibraryLocators`. The `LibraryLocators` are the plug-ins available with the type and subtype requested by the caller. The caller is required to dispose of the handle when it is done. The returned handle is `NULL` if there is an error or if there are no `LibraryLocators` of the type and subtype.

Calling `psGetSubtypeLocators` causes the Printing Plug-ins Manager to determine if its cache of plug-ins for the type and subtype specified by the call is current. If it is not, the cache is updated. Once the cache is up to date, the Printing Plug-ins Manager returns the number of locators matching the specified type and subtype in `*numLocatorsP`. If the number of `LibraryLocators` matching the type, subtype is greater than zero, a handle that points to the list of matching `LibraryLocators` is returned in `*locatorListHP`. It is up to the caller of `psGetSubtypeLocators` to dispose of this handle.

If the list contains more than one `LibraryLocator` the caller must determine which of the `LibraryLocators` in the list is of interest. Note that `LibraryLocators` that correspond to libraries in the Printing Plug-ins Folder are returned first (external plug-ins have precedence in this list). If a client needs more information to determine which library is the one to use, it should probably use the more complex call `psGetPreferredSubtypeLocators` described below.

**Note:**

The subtype `'zsys'` is reserved for use by the Printing Plug-ins Manager and cannot be used by a client. If a client calls `psGetSubtypeLocators` with a subtype of `'zsys'`, an `errInvalidPluginSubType` is returned.

## More Complex Case

```
OSStatus psGetPreferredSubtypeLocators(SettingsDataType type,
    SettingsDataSubType subtype, SInt32 clientDataVersion,
    ClientBuildCacheDataCallBackUPP buildcallback,
    ClientProcessCacheDataCallBackUPP processcallback,
    void *clientrefcon, long *numLocatorsP,
    LibraryLocatorListH *locatorListHP);

typedef pascal OSStatus (*ClientBuildCacheDataCallBack)
    (const LibraryLocator *thisLocator,
    void *clientrefcon, CacheData cache);

typedef pascal OSStatus (*ClientProcessCacheDataCallBack)
    (const LibraryLocator *thisLocator,
    void *clientrefcon, SInt32 size, const void *dataPtr,
    Boolean *addThisLocator);

OSStatus psAddSettingsSubTypeData(CacheData cache,
    SettingsDataSubType subtype, SInt32 size,
    const void *dataPtr);
```

The `psGetPreferredSubtypeLocators` routine is available when a client has more complex needs. This is the case if a plug-in of a given type can handle multiple subtypes. In this situation, there is some additional work that a client must do to categorize the subtypes that such a plug-in can handle. A second situation where the `psGetPreferredSubtypeLocators` call is useful is if there is data associated with the plug-in type and subtype that

is used to further refine which plug-in is the needed one. This might be the case if there are multiple plug-ins of a given type and subtype.

For example, each Download Manager plug-in can potentially handle multiple subtypes. Further, the Download Manager may have multiple plug-ins that can handle a given subtype of data, and there is data about each plug-in that the Download Manager uses to narrow the list of plug-ins which are available for a given download. This data can be stored in the Printing Plug-ins Manager's cache and used to process the `psGetPreferredSubtypeLocators` call.

A client calls `psGetPreferredSubtypeLocators` with `ClientBuildCacheDataCallBack` and `ClientProcessCacheDataCallBack` procedures. If the plug-in manager cache for this type needs to be rebuilt (which it does the first time a given type is seen and later when the cache becomes invalid), the `ClientBuildCacheDataCallBack` procedure is called for each library of that type. Once the cache and cached data are up to date, the Printing Plug-ins Manager calls the supplied `ClientProcessCacheDataCallBack` to examine the results.

The `ClientBuildCacheDataCallBack` procedure is responsible for gathering its data from each library (e.g., the Download Manager calls into the library to obtain the data it needs) and it should call the `psAddSettingsSubTypeData` routine for each subtype for which it has data to add to the cache. For each `psAddSettingsSubTypeData` call made, the Printing Plug-ins Manager *replaces* any data which may have already been added to its cache for this library and subtype. Note that the plug-in subtype for a library passed to the `ClientBuildCacheDataCallBack` procedure may not correspond to the subtype passed in to the `psGetPreferredSubtypeLocators` call since the process of rebuilding the cache processes each library of a given type.

Once the cache has been updated, the Printing Plug-ins Manager calls the `ClientProcessCacheDataCallBack` procedure for each `LibraryLocator` which supports the subtype requested by the `psGetPreferredSubtypeLocators` call. This gives the client an opportunity to use the data the `ClientBuildCacheDataCallBack` has cached for each `LibraryLocator` to pick the appropriate `LibraryLocator`. The `ClientProcessCacheDataCallBack` can make further calls to the shared library corresponding to that `LibraryLocator` in order to aid in its processing. Once a given `LibraryLocator` has been processed, the `ClientProcessCacheDataCallBack` routine returns a Boolean indicating whether to keep this `LibraryLocator` in its list of `LibraryLocators` to return from `psGetPreferredSubtypeLocators` in the `locatorListHP` variable.

For example, the Download Manager uses the `psGetPreferredSubtypeLocators` call. Download Manager plug-ins all have the subtype '????' because they potentially each handle multiple subtypes of data. In its `ClientBuildCacheDataCallBack` routine, the Download Manager obtains information about what type of data a given Download Manager converter library can handle and it supplies that data as its data to cache for that library. This data is cached by the Printing Plug-ins Manager so that further calls to `psGetPreferredSubtypeLocators` do not call the `ClientBuildCacheDataCallBack` routine unless the set of Printing Plug-ins has changed. Once the cache is up to date, the `psGetPreferredSubtypeLocators` calls the supplied `ClientProcessCacheDataCallBack` routine.

In its `ClientProcessCacheDataCallBack` routine, the Download Manager uses the cached information about what type of data a given converter library can handle to determine if it needs to call the library itself for more information. For each library which can handle a given type of data, the Download Manager loads and calls that library to refine the search further.

The parameter `clientDataVersion` passed to the `psGetPreferredSubtypeLocators` call is a number supplied by the caller and is used to put a version tag on the data the client puts in the cache. Since the client may change over a period of time, it is important to ensure that a new client doesn't receive cached data that was created by an older version of the client. A client only needs to change its version data when the client's custom cached data type changes.

**Warning:**

It is not anticipated that a large number of developers will use the `psGetPreferredSubtypeLocators` call to manage plug-ins of the same type. It is important that each client of this call with a given type ensure that the values of `clientDataVersion` it uses are unique. Failing to do so could potentially generate collisions in any cached data between clients and possibly return incorrect data to a given client. Following a simple guideline of choosing a large random number for your `clientDataVersion` will satisfy the need to avoid collisions between clients. Small integers are greatly discouraged.

**Note:**

Negative version numbers are reserved by the implementation and must not be used by clients. In later versions of SettingsLib, attempting this will generate the error `errInvalidPluginsClientDataVersion`.

The parameter `clientrefcon` passed to the `psGetPreferredSubtypeLocators` call is a pointer to client-supplied data. The `clientrefcon` supplied to this routine is typically a pointer to data that the client is gathering. The `clientrefcon` supplied to the `psGetPreferredSubtypeLocators` will be the `clientrefcon` passed to the `ClientBuildCacheDataCallBack` and `ClientProcessCacheDataCallBacks`.

Upon return from the `psGetPreferredSubtypeLocators` call, `*numLocatorsP` contains the number of `LibraryLocators` for which the `ClientProcessCacheDataCallBack` returns true in the `*addThisLocator` variable. Additionally, the `*locatorListHP` variable contains a handle which is a list of the `LibraryLocators` for which the `ClientProcessCacheDataCallBack` returned true in the `*addThisLocator` variable. If the returned value of `*numLocatorsP` is zero, the value of `*locatorListHP` will be NULL. If the returned value of `*numLocatorsP` is non-zero, it is up to the caller of `psGetPreferredSubtypeLocators` to dispose of the handle returned in `locatorListHP`.

**Note:**

The subtype `'zsys'` is reserved for use by the Printing Plug-ins Manager and cannot be used by a client. If a client calls `psGetSubtypeLocators` with a subtype of `'zsys'`, the error `errInvalidPluginSubType` is returned.

[Back to top](#)

## Utility Routines

There are several utility routines that may be helpful to clients of the Printing Plug-ins Manager.

### `psValidateLibraryLocator`

```
OSStatus psValidateLibraryLocator(
    const LibraryLocator *libLocatorP, Boolean *isValid);
```

The `psValidateLibraryLocator` allows a client to have the Printing Plug-ins Manager validate any `LibraryLocator`. If the data in the `LibraryLocator` pointed to by `libLocatorP` is valid, then `*isValid` is set to true; otherwise `*isValid` is set to false. `LibraryLocators` returned by the `psGetSubtypeLocators` and

`psGetPreferredSubtypeLocators` calls are always valid at the time they are returned, but they can become invalid because of user actions. If `LibraryLocators` are stored or provided to other pieces of software, it is important to validate them with the `psValidateLibraryLocator` routine before using them. For example, if the user updates a library and that library is referenced by a stored `LibraryLocator`, the `LibraryLocator` may no longer contain correct information about the stored shared library, and use of it with `GetDiskFragment` may cause a crash.

### `psGetPrintingPluginsFolder`

```
OSStatus psGetPrintingPluginsFolder(short *vref, long *folderId);
```

The `psGetPrintingPluginsFolder` call allows a client to obtain the volume reference number and `folderID` information for the folder that is being used as the Printing Plug-ins Folder. This is useful to clients such as installers who don't want to use the other Printing Plug-ins Manager calls to manage plug-in information but do want to know where the Printing Plug-ins Folder is.

### `psGetPrintingPluginsFolderName`

```
StringPtr psGetPrintingPluginsFolderName(void);
```

The `psGetPrintingPluginsFolderName` call allows a client to obtain the name of the folder that is being used as the Printing Plug-ins Folder. This is useful to clients such as installers who don't want to use the other Printing Plug-ins Manager calls to manage plug-in information but do want to know the name of the folder. The `StringPtr` returned by this call is only valid while the `SettingsLib` library is open.

### `psGetPluginPrefsFileName`

```
StringPtr psGetPluginPrefsFileName();
```

The `psGetPluginPrefsFileName` call allows a client to obtain the name for the file that is being used as the preferences file by the Printing Plug-ins Manager. This call is currently used only by the Printing Plug-ins Manager. There should be no need for third parties to access data in the preferences file and all data in that file is considered private.

### `psGetTypeLibraryData`

```
OSStatus psGetTypeLibraryData(SettingsDataType type,
    long *numLibrariesP, LibraryDataListH *libraryListHP);
```

`psGetTypeLibraryData` returns a list of `LibraryData` structures for all the plug-ins of the specified type. The `LibraryData` structure is defined as:

```
typedef struct LibraryData{
    SettingsDataSubType subtype;
    LibraryLocator libraryLocator;
}LibraryData, *LibraryDataListP, **LibraryDataListH;
```

The subtype field of the `LibraryData` structure is the subtype of the plug-in library represented by the `libraryLocator` field of the `LibraryData` structure.

A client calls this routine to enumerate all plug-ins of a given type. This call returns a handle containing a list of `LibraryData` structures. The `LibraryLocators` are all those plug-ins available with the type specified by the caller. The caller is required to dispose of the returned handle when it is done with it. The returned handle is `NULL` if there is an error or if there are no plug-ins of the type specified.

Note that `LibraryLocators` which correspond to libraries in the Printing Plug-ins Folder are returned first; i.e., external plug-ins have precedence in this list.

Use `psGetSubtypeLocators` instead of `psGetTypeLibraryData` when the subtype of the desired plug-in is known. The `psGetSubtypeLocators` call caches data to improve performance whereas `psGetTypeLibraryData` always looks at all the libraries in PrintingLib and the Printing Plug-ins Folder to obtain its list of libraries.

**Note:**

The `psGetTypeLibraryData` call is only available in PrintingLib version 8.6.5 and later. Clients should weak-link to PrintingLib and test that the symbol is available before calling `psGetTypeLibraryData`. For example:

```
if(psGetTypeLibraryData != (void *)kUnresolvedCFragSymbolAddress){
    psGetTypeLibraryData(...)
}else{
    // alternative approach ...
}
```

[Back to top](#)

## Errors

### `errInvalidPluginSubType`

Returned by the Printing Plug-ins Manager if the requested subtype is an invalid subtype. The only invalid subtype is the reserved subtype `'zsys'`.

### `errInvalidPluginsCache`

Even though the Printing Plug-ins Manager validated its cache before using it, the cache was not valid. This should never happen, but there is explicit checking to return this error if it does.

### `errInvalidPluginsClientDataVersion`

This error is returned if a caller of `psGetPreferredSubtypeLocators` passes in an invalid value for the argument `clientDataVersion`. Negative versions are reserved by the implementation and cannot be used by clients.



[Back to top](#)

## Summary

The Printing Plug-ins Manager is used by the Download Manager supplied with LaserWriter 8 Version 8.6 to manage Printing Plug-ins, such as low-level converters, custom hoses and Desktop Printer Utility plug-ins. Developers may wish to use it for writing plug-ins for LaserWriter 8, such as low-level converters, which will be described in a future Technote. The APIs presented in this Technote are also needed to install a printing plug-in correctly.

## References

[Technote 1169: The Download Manager](#)[Technote 1165: Introducing the LaserWriter 8 Driver Version 8.6.5](#)[Technote 1144: Writing Custom Hoses for LaserWriter 8.6](#)[Technote 1143: Introducing the LaserWriter 8 Driver Version 8.6](#)[Technote 1113: Customizing Desktop Printer Utility](#)[Back to top](#)

## Change History

01-April-1998	Originally written.
01-January-1999	Revised.
24-May-1999	Revised yet again.

[Back to top](#)

## Downloadables



Acrobat version of this Note (K).

[Download](#)

Binhexed Routine Descriptor Lib (147K).

[Download](#)[Back to top](#)