Table of Contents

Chapter

1

Install

Use the Install object to manage the downloading and installation of software with the XPI Installation Manager.

Install Overview

The Install object is used primarily in installation scripts, and less often in trigger scripts on web pages. In all cases, the `Install` object is implicit—like the `window` object in regular web page scripts—and needn't be prefixed to the object methods. The following two lines, for example, are equivalent:

```
f = getFolder("Program");
f = Install.getFolder("Program");
```

An installation script is composed of calls to the Install object, and generally takes the following form:

Initialize the installation	call *initInstall* with the name of the installation and the necessary registry and version information.
Add the files to the installation	Add files to the installation by calling *getFolder* to get file objects and passing those object refs to *addFile* as many times as necessary.

Perform installation	Check that the files have been added successfully (e.g., by checking the error *Return Codes* from many of the main installation methods, and go ahead with the install if everything is in order:
	`performOrCancel();`
	`function performOrCancel()`
	`{`
	`  if (0 == getLastError())`
	`  performInstall();`
	`  else`
	`  cancelInstall();`
	`}`

For complete script examples, see *Script Examples.*

Method Reference

addDirectory	Unpacks an entire subdirectory.
addFile	Unpacks a single file.
alert	Displays an Alert dialog box with a message and an OK button.
cancelInstall	Aborts the installation of the software.
confirm	Displays a Confirm dialog box with the specified message and OK and Cancel buttons.
deleteRegisteredFile	Deletes the specified file and its entry in the Client Version Registry.
execute	Extracts a file from the XPI file to a temporary location and schedules it for later execution.
gestalt	Retrieves information about the operating environment. (Mac OS only)
getComponentFolder	Returns an object representing the directory in which a component is installed.
getFolder	Returns an object representing a directory, for use with the `addFile` method.
getLastError	Returns the most recent non-zero error code.
getWinProfile	Constructs an object for working with a Windows .ini file.
getWinRegistry	Constructs an object for working with the Windows Registry.
initInstall	Initializes installation for the given software and version.
loadResources	Returns an object whose properties are localized strings loaded from the specified property file.
logComment	Add a comment line to the install log.
patch	Applies a set of differences between two versions.
performInstall	Finalizes the installation of the software.
registerChrome	Registers chrome with the chrome registry.
resetError	Resets a saved error code to zero.
setPackageFolder	Sets the default package folder that is saved with the root node.

addDirectory

Unpacks an entire directory into a temporary location.

Method of

`Install`

Syntax

```
public int addDirectory (
   String xpiSourcePath);
```

```
public int addDirectory (
   String registryName,
   String xpiSourcePath,
   Object localDirSpec,
   String relativeLocalPath);
```

```
public int addDirectory (
   String registryName,
   String version,
   String xpiSourcePath,
   Object localDirSpec,
   String relativeLocalPath);
```

```
public int addDirectory (
   String registryName,
   String version,
   String xpiSourcePath,
   Object localDirSpec,
   String relativeLocalPath,
   Boolean forceUpdate);
```

```
public int addDirectory (
   String registryName,
   InstallVersion version,
   String xpiSourcePath,
   Object localDirSpec,
   String relativeLocalPath,
   Boolean forceUpdate);
```

Parameters

The `addDirectory` method has the following parameters:

`registryName`	The pathname in the Client Version Registry for the root directory of the files that are to be installed. This parameter can be an absolute pathname (beginning with a /) or a relative pathname, (not beginning with a slash). An absolute pathname is used as specified. A relative pathname is appended to the registry name of the package as specified by the `package` parameter to the *`initInstall`* method. This parameter can also be null, in which case the `xpiSourcePath` parameter is used as a relative pathname. Note that the registry pathname is not the location of the software on the computer; it is the location of information about the software inside the Client Version Registry.
`version`	An `InstallVersion` object or a `String` of up to four integer values delimited by periods, such as "1.17.1999.1517". For variants or this method without a version argument the value from `initInstall` will be used.
`xpiSourcePath`	A string specifying the location of the directory within the XPI file. An empty string ("") causes the creation of a subdirectory node in the registry without actually unpacking any files, which may be useful when you are updating a package that contains subcomponents that could also be updated separately. When `xpiSourcePath` is an empty string, `registryName` cannot be null.
`localDirSpec`	An object representing a directory. The directory is installed under this directory on the user's computer. You create this object by passing a string representing the directory to the `getFolder` method.
`subdir`	The name of a directory to append to `localDirSpec`, using `/' as the path separator regardless of the platform. If `subdir` is missing, `null`, or `""`, the filenames are appended directly to the folder name specified by `localDirSpec`.
`flags`	An optional field; reserved for future use. Pass 0 as the default value.

Returns

An integer error code. For a list of possible values, see *Return Codes.* In some situations, `addDirectory` may return other errors. For example, if the error was with regard to the signing of the XPI file, `addDirectory` returns a security error code.

Description

The `addDirectory` method puts the files in the specified directory in a temporary location. To move the files and all other subcomponents to their final location, call the `performInstall` method after you've successfully added all subcomponents.

addFile

Unpacks a single subcomponent into a temporary location. Queues the subcomponent for addition to the Client Version Registry and installation to its final destination.

Method of

`Install`

Syntax

```
public int addFile (
   String registryName,
   InstallVersion version,
   String xpiSourcePath,
   Object localDirSpec,
   String relativeLocalPath,
   Boolean forceUpdate);

public int addFile (
   String registryName,
   String version,
   String xpiSourcePath,
   Object localDirSpec,
   String relativeLocalPath,
   Boolean forceUpdate);

public int addFile (
   String xpiSourcePath);

public int addFile (
   String registryName,
   String xpiSourcePath,
   Object localDirSpec,
```

```
   String relativeLocalPath);

public int addFile (
   String registryName,
   String version,
   String xpiSourcePath,
   Object localDirSpec,
   String relativeLocalPath);
```

Parameters

The `addFile` method has the following parameters:

`registryName`	The pathname in the Client Version Registry about the file. This parameter can be an absolute pathname, such as `/royalairways/RoyalSW/executable` or a relative pathname, such as `executable`. Typically, absolute pathnames are *only* used for shared components, or components that come from another vendor, such as `/Microsoft/shared/msvcrt40.dll`. Typically, relative pathnames are relative to the main pathname specified in the `initInstall` method. This parameter can also be null, in which case the `xpiSourcePath` parameter is used as a relative pathname. Note that the registry pathname is not the location of the software on the machine; it is the location of information about the software inside the Client Version Registry.
`version`	An *`InstallVersion`* object or a `String` of up to four integer values delimited by periods, such as "1.17.1999.1517". For variants or this method without a version argument the value from *`initInstall`* will be used.
`xpiSourcePath`	A string specifying the location of the file within the XPI file.
`localDirSpec`	An object representing a directory. The file is installed under this directory on the user's machine. You create this object by passing a string representing the directory to the `getFolder` method.
`relativeLocalPath`	A pathname relative to the `localDirSpec` parameter. The file is installed in this location on the user's machine. You must always use forward slashes (/) in this pathname, regardless of the convention of the underlying operating system. If this parameter is blank or `NULL, xpiSourcePath` is used.
`flags`	An optional field; reserved for future use. Pass 0 as the default value.

Returns

An integer error code. For a list of possible values, see *Return Codes.*

Description

The `addFile` method puts the file in a temporary location. To move this and all other files to their final location, call the *performInstall* method after you've successfully added all files.

Example

```
var xpiSrc = "file.txt";

initInstall("Adding a File",
   "addFile",
   "1.0.1.7",
   1);

f = getFolder("Program");
setPackageFolder(f);
addFile(xpiSrc);

if (0 == getLastError())
   performInstall();
else
   cancelInstall();
```

alert

The alert function displays a model dialog box with a message representing the input.

Method of

```
Install
```

Syntax

```
void alert ( String string );
```

Parameters

Though it's most common to input a string for display in an alert dialog, the single input parameter for `alert()` can be a value of any data type. You can, for example, input an object reference and see that object displayed as a string in the alert dialog.

Returns

Nothing.

cancelInstall

Aborts installation of the software; performs cleanup of temporary files.

Method of

`Install`

Syntax

```
void cancelInstall()
void cancelInstall( int errorCode )
```

Parameters

None.

Returns

An integer error code. The optional argument is an error code that can be returned to the triggering page. For most purposes, it's recommended that one of the standard return codes be used. But a script can, in fact, return any valid integer. For a list of possible values, and any custom `errorCode` created by install writer, see *Return Codes*.

Example

Use the following code to abort or to finalize an installation, based on a variable you set earlier in your code:

```
initInstall("Royal Airways TripPlanner","/RoyalAirways/
TripPlanner","1.0.0.0");
...
err = getLastError();
if (!err)
   performInstall();
else
   cancelInstall(err);
```

confirm

Displays a modal confirmation dialog.

Method of

`Install`

Syntax

```
Boolean aReturn confirm( String string );
```

Parameters

The input parameter is the string to be displayed in the confirmation dialog. This string is typically in the form of a prompt for the user (e.g., “Are you sure you want to delete the selected file(s)?”).

Returns

The return value is a boolean indicating whether the user has selected “OK” (value=1) or the “Cancel” (value=0).

deleteRegisteredFile

(Netscape 6 and Mozilla do not currently support this method.)

Deletes the specified file and removes its entry from the Client Version Registry.

Method of

`Install`

Syntax

```
int deleteRegisteredFile
   (String registryName);
```

Parameters

The `deleteRegisteredFile` method has the following parameter:

`registryName`	The pathname in the Client Version Registry for the file that is to be deleted.

Returns

An integer error code. For a list of possible values, see *Return Codes.*

Description

The `deleteRegisteredFile` method deletes the specified file and removes the file's entry from the Client Version Registry. If the file is currently being used, the name of the file that is to be deleted is saved and Netscape 6 attempts to delete it each time it starts up until the file is successfully deleted. This method is used to delete files that cannot be removed by the `uninstall` method or to remove files that are no longer necessary or whose names have changed.

execute

Launches a file inside the install archive.

Method of

`Install`

Syntax

```
int execute (
   xpiSourcePath)
```

```
int execute (
   String xpiSourcePath,
   String args);
```

Parameters

The `execute` method has the following parameters:

`xpiSourcePath`	The pathname of the file to extract and execute.
`args`	A parameter string that is passed to the executable. (Ignored on Mac OS)

Returns

An integer error code. For a list of possible values, see *Return Codes.*

Description

The `execute` method extracts the named file from the XPI file to a temporary file name. Your code must call the *performInstall* method to actually execute the file. You can use this method to launch an InstallShield installer or any install executable file stored in a XPI file.

gestalt

(Macintosh only)

Retrieves information about the operating environment.

Method of

`Install`

Syntax

```
OSErr gestalt (
   String selector,
   long * response);
```

Parameters

The `gestalt` method takes the following parameters:

`selector`	The selector code for the information you want.
`response`	On return, the requested information. The format depends on the select code specified in the `selector` parameter.

Description

The `gestalt` method is a wrapper for the `gestalt` function of the Macintosh Toolbox. For information on that function, see *Inside Macintosh: Operating System Utilities.*

This method returns null on Unix and Windows platforms.

getComponentFolder

Returns an object representing the directory in which a component is installed.

Method of

`Install`

Syntax

```
Object getComponentFolder
   (String registryName)
```

```
Object getComponentFolder (
   String registryName,
   String subDirectory);
```

Parameters

The `getComponentFolder` method has these parameters:

`registryName`	The pathname in the Client Version Registry for the component whose installation directory is to be obtained.
`subDirectory`	A string that specifies the name of a subdirectory. If the specified subdirectory doesn't exist, it is created. This parameter is available in Netscape 6 and may be case sensitive (depending on the operating system).

Returns

An object representing the directory in which the component is installed, or `NULL` if the component could not be found or if `subDirectory` refers to a file that already exists.

Description

The `getComponentFolder` method to find the location of a previously installed software package. Typically, you use this method with the `addFile` method or the `addDirectory` method.

getFolder

Returns an object representing one of Netscape's standard directories.

Method of

`Install`

Syntax

```
FileSpecObject getFolder (
   String FolderName);

FileSpecObject getFolder (
   String folderName,
   String subDirectory);

FileSpecObject getFolder (
   Object localDirSpec,
   String subDirectory);
```

Parameters

The getFolder method has the following parameters:

`folderName`	A string representing one of Netscape's standard directories. There are two sets of possible values for this parameter. The first set contains platform-independent locations; the second set contains platform-specific locations. You are encouraged to use the platform-independent locations. See the list in the Description section for the two sets of locations.
`subDirectory`	A string that specifies the name of a subdirectory. If the specified subdirectory doesn't exist, it is created. This parameter is available in Netscape 6 or later and may be case sensitive (depending on the operating system).
`localDirSpec`	A `FileSpecObject` representing a directory obtained by *getComponentFolder* or *getFolder*.

Returns

A `FileSpecObject` representing one of Netscape's standard directories, or `NULL` in case of error or if `subDirectory` refers to a file that already exists.

Description

The `getFolder` method obtains an object representing one of Netscape's standard directories, for use with the `addFile` and `getWinProfile` methods.

The value of `folderName` must be one of the following:

Platform-independent locations	Platform-dependent locations
`"Chrome"`	`"Mac Apple Menu"`
`"Components"`	`"Mac Control Panel"`
`"Current User"`	`"Mac Desktop"`
`"Defaults"`	`"Mac Documents"`
`"file:///"`	`"Mac Extension"`
`"OS Drive"`	`"Mac Fonts"`
`"Plugins"`	`"Mac Shutdown"`
`"Preferences"`	`"Mac Startup"`
`"Program"`	`"Mac System"`
`"Temporary"`	`"Mac Trash"`
	`"Mac Preferences"`
	`"Unix Lib"`
	`"Unix Local"`
	`"Win System"`
	`"Windows"`

The "file:///" form is only valid when the `subDirectory` parameter is used. It must be in file: URL format minus the "file:///" part. For example,

```
mydir = getFolder("file:///", "cl/mysoftco/somedir);
```

Note that forward slashes are used, regardless of the platform.

The folders whose names start with "Win", "Mac", or "Unix" are specific to those platforms. You should be careful about using one of those directories, as it makes your installation platform-specific.

Example

To get an object representing the standard plug-ins directory, you would use this call:

```
plugindir = getFolder("Plugins");
```

getLastError

Returns the most recent nonzero error code.

Method of

`Install`

Syntax

```
int getLastError ();
```

Parameters

None.

Returns

The most recent nonzero error code. For a list of possible values, see *Return Codes*.

Description

Use `getLastError` method to obtain the most recent nonzero error code since `initInstall` or `resetError` were called. This method allows you to defer checking for error codes each time you call `addFile` or `addDirectory` until the last `addFile` or `addDirectory` call.

The `getLastError` method does not return errors from methods that return objects, such as `getFolder`.

Example

The following example calls `getLastError` after a series of `addFile` calls:

```
addFile("npplug", ...);
addFile("/MS/Shared/ctl3d.dll", ...);
addFile("/NetHelp/royalplug/royalhelp.html",...);
err = getLastError();
```

getWinProfile

(Windows only)

Constructs an object for working with a Windows .ini file.

Method of

`Install`

Syntax

```
WinProfile getWinProfile (
   Object folder,
   String file);
```

Parameters

The `getWinProfile` method has the following parameters:

`folder`	An object representing a directory. You create this object by passing a string representing the directory to the *getFolder* method.
`file`	A relative pathname to an initialization file in the directory specified by the `folder` parameter, such as `"royal/royal.ini"`.

Returns

A `WinProfile` object.

Description

The `getWinProfile` method creates an object for manipulating the contents of a Windows `.ini` file. Once you have this object, you can call its methods to retrieve strings from the file or to add strings to the file. For information on the returned object, see `WinProfile`.

This method returns null on Unix and Macintosh platforms.

Example

To edit the `win.ini` file, you would create a WinProfile object with this call:

```
getWinProfile (getFolder("Windows"), "win.ini");
```

getWinRegistry

(Windows only)

Constructs an object for working with the Windows Registry.

Method of

`Install`

Syntax

```
WinReg getWinRegistry();
```

Parameters

None.

Returns

A *WinReg* object.

Description

Use the `getWinRegistry` method to create an object for manipulating the contents of the Windows Registry. Once you have this object, you can call its methods to retrieve or change the registry's content. For information on the returned object, see *WinReg*.

This method returns `NULL` on Unix and Macintosh platforms.

initInstall

Initializes the installation of the specified software and version.

Method of

`Install`

Syntax

```
int initInstall (
   String displayName,
   String package,
   InstallVersion version,
   int flags);
```

```
int initInstall (
```

```
   String displayName,
   String package,
   String version,
   int flags);

int initInstall (
   String displayName,
   String package,
   String version);

int initInstall (
   String displayName,
   String package,
   InstallVersion version);
```

Parameters

The `initInstall` method has the following parameters:

`displayName`	A `string` that contains the name of the software being installed. The name of the software is displayed to the user.
`package`	The Client Version Registry pathname for the software (for example: `Plugins/Adobe/Acrobat or /royalairways/RoyalPI/`). It is an error to supply a null or empty name. The name can be absolute or relative. A relative pathname is relative to the Netscape 6 namespace. A relative pathname must start with `plugins/`, to be relative to the plug-ins portion of that namespace or `java/download/`, to be relative to the Java portion. All other parts of the Netscape 6 area of the registry are reserved for use by Netscape. The Client Version Registry is a hierarchical description of the software registered for use with Netscape 6. The registry name provided here is not the location of the software on the machine, it is the location of information about the software inside the registry. This distinction is important when you add components with the `addFile` method.
`version`	An *InstallVersion* object or a `String` representing the version of the package being installed. When `version` is a `String` it should be up to four integer values delimited by periods, such as "1.17.1999.1517".
`flags`	An optional field; reserved for future use. Pass 0 as the default value.

Returns

An integer error code. For a list of possible values, see *Return Codes.*

Description

The `initInstall` method initializes the installation of the specified software. You must call this method immediately after the constructor. It is an error to call any other `Install` methods before calling `initInstall`.

After calling `initInstall`, your script must call *performInstall* or *cancelInstall* before it finishes. If your script does not call `performInstall` or `cancelInstall`, Netscape 6 will not be able to clean up properly after your script finishes.

Example

To start installation for the Royal Airways plug-in, you could use this code:

```
initInstall("Royal Airways TripPlanner","/RoyalAirways/
TripPlanner","1.0.0.0");
...
err = getLastError();
if (!err)
   performInstall();
else
   cancelInstall(err);
```

loadResources

Loads a properties file.

Method of

`Install`

Syntax

```
Object loadResources( String xpiPath );
```

Parameters

The sole input parameter for `loadResources` is a string representing the path to the properties file *in the XPI* in which the key/value pairs are defined.

Returns

A JavaScript object whose property names are the keys from that file and whose values are the strings.

Description

The format of the properties file expected by `loadResources` is the same as that of the chrome locale .properties files. This method is used to internationalize installation scripts by allowing the installer to retrieve localized string values from a separate file. Be aware that the parameter specifies the file within the XPI and not on the file system, as the following example demonstrates.

Example

Given a XPI with this internal structure:

```
install.js
bin/res_eg_2.properties
bin/somefile.exe
bin/...
```

The following lines retrieve the properties as a JavaScript object and make the values accessible with the familiar “dot property” syntax:

```
resEg2Obj = loadResources("bin/res_eg_2.properties");
dump( resEg2.Obj.title )
```

logComment

Adds a comment line to the installation log.

Method of

`Install`

Syntax

```
int logComment( String aComment );
```

Parameters

The sole input parameter is a string whose value will be written to the log during the installation process.

Returns

An integer error code. For a list of possible values, see *Return Codes.*

Description

The install log is created in the product directory by default (where the browser executable is) if it can be, and if the installation doesn't have proper permission, the install log is written to the user's profile directory. Respectively, these directories correspond to the “Program” and “Current User” keywords for the *getFolder* method.

patch

Updates an existing component.

Method of

`Install`

Syntax

```
int patch (
   String registryName,
```

```
   String xpiSourcePath,
   Object localDirSpec,
   String relativeLocalPath);

int patch (
   String registryName,
   InstallVersion version,
   String xpiSourcePath,
   Object localDirSpec,
   String relativeLocalPath);

int patch (
   String registryName,
   String version,
   String xpiSourcePath,
   Object localDirSpec,
   String relativeLocalPath);
```

Parameters

The `patch` method has the following parameters:

`registryName`	The pathname in the Client Version Registry for the component that is to be patched. This parameter can be an absolute pathname, such as `/royalairways/RoyalSW/executable` or a relative pathname, such as `executable`. Typically, absolute pathnames are *only* used for shared components, or components that come from another vendor, such as `/Microsoft/shared/msvcrt40.dll`. Typically, relative pathnames are relative to the main pathname specified in the `initInstall` method. This parameter can also be null, in which case the `xpiSourcePath` parameter is used as a relative pathname. Note that the registry pathname is not the location of the software on the computer; it is the location of information about the software inside the Client Version Registry.
`version`	An *InstallVersion* object or a `String` of up to four integer values delimited by periods, such as "1.17.1999.1517". For variants or this method without a version argument the value from `initInstall` will be used.
`xpiSourcePath`	A string specifying the location of the differences file within the XPI file.
`localDirSpec`	An object representing the directory in which the subcomponent that is to be patched resides. You create this object by passing a string representing the directory to the `getFolder` method.
`relativeLocalPath`	A pathname relative to the `localDirSpec` parameter that identifies the subcomponent that is to be patched. You must always use forward slashes (/) in this pathname, regardless of the convention of the underlying operating system. If this parameter is blank or `NULL, xpiSourcePath` is used.

Returns

An integer error code. For a list of possible values, see "Return Codes" on page 102.

Description

The `patch` method to update an existing component by applying a set of differences between two known versions. The set of differences is in `GDIFF` format and is created by the `NSDiff` utility.

A patch can only be applied between two known versions. If the existing version of the file does not match the checksum stored in the `GDIFF` file, `patch` returns an error without applying the patch. After `patch` applies a patch, it compares a checksum.of the resulting file against a checksum stored in the `GDIFF` file. If the checksums do not match, the original version of the file is preserved, the patched version of the file is discarded, and an error code is returned.

Any single installation process can apply multiple patches to the same file.

If `performInstall` indicates that a reboot is necessary to complete the installation, `patch` may not work in subsequent XPInstall processes until the reboot is performed.

performInstall

Performs the acutal installation of the software. Moves all components to their final locations, launches any pending executions, and registers the package and all of its subcomponents in the Client Version Registry.

Method of

`Install`

Syntax

```
int performInstall();
```

Parameters

None.

Returns

An integer error code. For a list of possible values, see *Return Codes.* In some situations the method may return other errors. For example, if the error was with regard to the signing of the XPI file, it returns a security error. In a few cases you may get a registry error.

Example

Use the following code to abort or to finalize an installation, based on a variable you set earlier in your code:

```
initInstall("Royal Airways TripPlanner",
   "/RoyalAirways/TripPlanner",
   "1.0.0.0");
...
err = getLastError();
if (!err)
   performInstall();
else
   cancelInstall(err);
```

registerChrome

Registers chrome with the chrome registry.

Method of

`Install`

Syntax

```
int registerChrome(
  SWITCH,
  srcDir,
  xpiPath);
```

Parameters

The `patch` method has the following parameters:

`SWITCH`	`SWITCH` is the chrome switch indicating what type of file is being registered. `SKIN` is used to register skins, `LOCALE` is used to register language packs. `PACKAGE`, a third possibility is the equivalent of `SKIN AND/OR LOCALE`, and ensures that everything in the XPI is registered. One final option for the switch parameter is `DELAYED_CHROME`, which registers the switch only after a relaunch of the browser. Note that you can combine switches as in the example below.
`srcDIR`	`srcDIR` is a `FileSpecObject` representing the source destination of the installation. FileSpecObjects like that required by this function are created using the *getFolder* method on the Install object.
`xpiPath`	`xpiPath` is the path withing the XPI in where the contents.rdf file defining the chrome is located. “locale/myLocale/aim,” for exampel, points to the `locale/myLocale/aim` subdirectory of the same XPI file in which the installation script is located.

Returns

An integer error code. For a list of possible values, see *Return Codes.* In some situations the method may return other errors. For example, if the error was with regard to the signing of the XPI file, it returns a security error. In a few cases you may get a registry error.

Description

When the third parameter is omitted (pointing to a specific location within the XPI file), this function is being used in a somewhat deprecated way. In this case, `registerChrome` is supporting the old format of installation archives, in which the *manifest.rdf* file was always located at the highest level of the installation archive. In this case, registerChrome does not require a path inside the archive, as it does now in order to locate the more flexible *contents.rdf* format of installation archives.

Note that you can also look in the installed-chrome.txt file in the chrome directory to see how and where the `registerChrome` function has registered your package with the chrome registry.

Example

From one of the browser installation files, in which the main communicator archive (browser.xpi) is registered:

```
registerChrome(
  CONTENT | DELAYED_CHROME,
  getFolder(cf,"browser.xpi"),
  "content/editor/");
```

resetError

Resets a saved error code to zero.

Method of

`Install`

Syntax

```
void resetError ();
```

Parameters

None.

Returns

Nothing.

Description

The `resetError` method resets any saved error code to zero. See *getLastError* for additional information.

Example

To reset the last error code to zero:

```
resetError();
```

setPackageFolder

Sets the default package folder.

Method of

`Install`

Syntax

```
void setPackageFolder (
   Object folder);
```

Parameters

The `setPackageFolder` method has the following parameter:

`folder`	An object representing a directory. You create this object by passing a string representing the directory to the `getFolder` or `getComponentFolder` method.

Returns

None.

Description

The `setPackageFolder` method to set the default package folder that is saved with the root node. When the package folder is set, it is used as the default for forms of `addFile` and other methods that have an optional `localDirSpec` parameter.

You should only call this method once, and you should always call it immediately after you call `initInstall`. If you call `setPackageFolder` multiple times, the last folder set is the folder that is saved in the Client Version Registry and used as the default for other installations.

Chapter

2

InstallTrigger

A trigger script on a web page uses an `InstallTrigger` object to download and install software.

InstallTrigger Overview

For very simple installations, the install methods on the InstallTrigger object may be all that's needed in the installation script. For more complex installations, you may need to use the *Install* and/or *File* installation objects. In either case, you must trigger the installation process by creating a web page script in which InstallTrigger methods download the specified XPI file and "trigger" the execution of the install.js script at the top level of that XPI.

The principal method on the InstallTrigger object is *install*, which downloads and installs one or more software packages archived in the XPI file format. The following is a basic example of an install trigger on a web page:

```
xpi={'XPInstall Name':'simple.xpi'};
InstallTrigger.install(xpi);
```

You can also use the InstallTrigger object to check software versions, install Netscape 6/Mozilla skins and language packs, and perform multiple-package installations with *install*.

Method Reference

compareVersion	Compares the version of a file or package with the version of an existing file or package.
enabled	Indicates whether or not the Software Installation is enabled for this client machine.
getVersionInfo	Returns an *InstallVersion* object representing the version number from the Client Version Registry for the specified software or component.
install	Installs one or more XPI files on the local machine.
installChrome	Installs new skin or locale packages in Netscape 6 and Mozilla.
startSoftwareUpdate	Initiates the downloading and installation of the specified software. Note: Deprecated in favor of *install*.

compareVersion

Compares the version of a file or package with the version of an existing file or package.

Method of

`InstallTrigger`

Syntax

```
int compareVersion (
   String registryName,
   InstallVersion version);

int compareVersion (
   String registryName,
   String version);

int compareVersion (
   String registryName,
   int major,
   int minor,
   int release,
```

```
   int build);
```

Parameters

The `compareVersion` method has the following parameters:

`registryName`	The pathname in the Client Version Registry for the component whose version is to be compared. This parameter can be an absolute pathname, such as `/royalairways/RoyalSW` or a relative pathname, such as `plugsin/royalairway/RoyalSW`. Note that the registry pathname is not the location of the software on the computer; it is the location of information about the software inside the Client Version Registry.
`version`	An *`InstallVersion`* object containing version information or a `String` in the format *major.minor.release.build*, where *major*, *minor*, *release*, and *build* are integer values representing version information.

Returns

If the versions are the same or if Software Installation is not enabled, this method returns 0. If the version of `registryName` is smaller (earlier) than `version`, this method returns a negative number. Otherwise, this method returns a positive number. In particular, this method returns one of the following values:

- -4: `registryName` has a smaller (earlier) major number than `version`.
- -3: `registryName` has a smaller (earlier) minor number than `version`.
- -2: `registryName` has a smaller (earlier) release number than `version`.
- -1: `registryName` has a smaller (earlier) build number than `version`.
- 0: The version numbers are the same; both objects represent the same version.
- 1: `registryName` has a larger (newer) build number than `version`.
- 2: `registryName` has a larger (newer) release number than `version`.
- 3: `registryName` has a larger (newer) minor number than `version`.
- 4: `registryName` has a larger (newer) major number than `version`.

The following constants can be used to check the value returned by `compareVersion`:

```
int MAJOR_DIFF = 4;
int MINOR_DIFF = 3;
int REL_DIFF = 2;
int BLD_DIFF = 1;
int EQUAL = 0;
```

In Communicator 4.5, the following constants are defined and available for checking the value returned by `compareVersion`:

```
InstallTrigger.MAJOR_DIFF
InstallTrigger.MINOR_DIFF
InstallTrigger.REL_DIFF
InstallTrigger.BLD_DIFF
InstallTrigger.EQUAL
```

Description

The `compareVersion` method compares the version of an installed file or package with a specified version. It is often used as a check against which to initiate the installation process.

If `registryName` is not found in the Client Version Registry or if `registryName` does not have version, `registryName` is assumed to have a version of 0.0.0.0.

If `registryName` represents a file, `compareVersion` checks for the existence of the file. If the file has been deleted, its version is assumed to be 0.0.0.0.

enabled

Indicates whether or not Software Installation is enabled for this client machine.

Method of

`InstallTrigger`

Syntax

```
Boolean enabled ();
```

Parameters

None

Returns

True if Software Installation is enabled for this client machine; otherwise, false. The method reflects the value of the Software Installation preference in the user interface, and of the `xpinstall.enabled` preference in *pref.js*.

Example

The following code uses the *startSoftwareUpdate* method to unconditionally trigger a download from `http://royalairways/royalpkg.xpi` as long as Software Installation is enabled on the browser:

```
if (InstallTrigger.enabled() )
   InstallTrigger.startSoftwareUpdate ("http://royalair.com/
rasoft.xpi");
```

getVersionInfo

Returns an object representing the version number from the Client Version Registry for the specified component. It is used in both trigger scripts and installation scripts.

Method of

`InstallTrigger`

Syntax

```
InstallVersion getVersionInfo (
   String component );
```

Parameters

The `getVersionInfo` method has one parameter:

`component`	The name of a component in the Client Version Registry.

Returns

If Software Installation is disabled, this method returns `NULL`.

Otherwise, it returns an *InstallVersion* object representing the version of the component. If the component has not been registered in the Client Version Registry or if the specified component was installed with a null version, this method returns null.

Installing a component with a null version indicates that the component should always be updated when the opportunity arises.

install

Installs one or more XPI files on the local machine.

Method of

`InstallTrigger.`

Syntax

```
int install(array XPIlist [, function callBackFunc ] )
```

Parameters

The `install` method has the following parameters:

`XPIlist`	An array of files to be installed (see example below).
`callBackFunc`	An optional callback function invoked when the installation is complete (see example below).

Returns

`install` returns True if the function succeeded and False if it did not, but these values are not always reliable as a determinant of the success of the operation. To surface detail about the status of the installation, use the optional callback function and its status paramter, as in the example below.

Description

In the example below, a special JavaScript object constructor is used to create an object that can be passed to the `install()` method. The `{ }` constructor takes a comma-delimited set of label/value pairs. For installations, these pairs are the name and the path of the XPI, respectively. In the example below, a single installation object is created, but you can use this approach to create multiple installations to pass to a single `install`.

As with the older *startSoftwareUpdate* method, XPIs installed with this method must have their own install.js files in which the full installation is defined. In contrast to *startSoftwareUpdate*, `install` allows you to do multiple installs with the same trigger.

Example

```
function xpinstallCallback(url, status)
{
if (status == 0)
   msg = "XPInstall Test:   PASSED\n";
else
   msg = "XPInstall Test:   FAILED\n";

   dump(msg);
   alert(msg);
}
```

```
xpi={'XPInstall Pre-Checkin Test':'pre_checkin.xpi'};
InstallTrigger.install(xpi,xpinstallCallback);
```

installChrome

Installs new skin or locale packages in Netscape 6 and Mozilla.

Method of

`InstallTrigger`

Syntax

```
int installChrome( TYPE, url, name )
```

Parameters

The `installChrome` method has the following parameters:

`TYPE`	`TYPE` can be `InstallTrigger.SKIN` or `InstallTrigger.LOCALE`.
`url`	url is a string containing a full or relative URL to download
`name`	name is displayed in the dialog, but is also used to *select* the theme so must match exactly the name in the internal manifest.rdf file.

Returns

A boolean value indicating False if the software install feature has been turned off, and True if it's on. Note that this return value does not indicate anything about the success of the installation.

Description

`installChrome` is a special method for installing new chrome in Netscape 6 and Mozilla. The method performs a simplified installation of language packs or Netscape 6/Mozilla skins, and saves you the trouble of writing separate installation scripts in the XPI files or using the more sophisticated methods of the *Install* and *File* objects.

startSoftwareUpdate

Triggers the downloading and installation of the software at the specified URL.

Method of

`InstallTrigger`

Syntax

```
Boolean startSoftwareUpdate (
   String url,
   int flag);
```

```
Boolean startSoftwareUpdate (
   String url);
```

Parameters

The `startSoftwareUpdate` method has the following parameters:

`url`	A uniform resource locator specifying the location of the XPI file containing the software.
`flag`	An optional field; reserved for future use. Pass 0 as the default value.

Returns

True.

Description

The `startSoftwareUpdate` method triggers a software download and install from the specified URL. This method has been largely superceded by newer *install* method, which is more flexible and allows you to install more than one XPI.

Note also that XPIs installed with this method must have their own *install.js* files in which the full installation is defined.

Chapter

3

InstallVersion

You use `InstallVersion` objects to contain version information for software.

InstallVersion Overview

This object and its methods are used both when triggering a download, to see whether a particular version needs to be installed, and when installing the software.

Method Reference

`compareTo`	Compares the version information specified in this object to the version information specified in the `version` parameter.
`init`	Initialize an `InstallVersion` object.

compareTo

Compares the version information specified in this object to the version information specified in the `version` parameter.

Method of

`InstallVersion`

Syntax

```
compareTo (
   InstallVersion version);

compareTo (
   String version);

compareTo (
   int major,
   int minor,
   int release,
   int build);
```

Parameters

The `compareTo` method has the following parameters:

`maj`	The major version number.
`min`	Minor version number.
`rev`	Revision number.
`bld`	Build number.
`version`	An `InstallVersion` object or a `String` representing version information in the format "4.1.2.1234".

Returns

If the versions are the same, this method returns 0. If this version object represents a smaller (earlier) version than that represented by the `version` parameter, this method returns a negative number. Otherwise, it returns a positive number. In particular, this method returns one of the following values:

- -4: This version object has a smaller (earlier) major number than `version`.
- -3: This version object has a smaller (earlier) minor number than `version`.
- -2: This version object has a smaller (earlier) release number than `version`.
- -1: This version object has a smaller (earlier) build number than `version`.
- 0: The version numbers are the same; both objects represent the same version.
- 1: This version object has a larger (newer) build number than `version`.
- 2: This version object has a larger (newer) release number than `version`.
- 3: This version object has a larger (newer) minor number than `version`.
- 4: This version object has a larger (newer) major number than `version`.

The following constants are defined and available for checking the value returned by `compareTo`:

```
InstallVersion.MAJOR_DIFF
InstallVersion.MINOR_DIFF
InstallVersion.REL_DIFF
InstallVersion.BLD_DIFF
InstallVersion.EQUAL
```

Example

This code uses the `compareTo` method to determine whether or not version 3.2.1 of the Royal Airways software has been previously installed:

```
existingVI = InstallTrigger.getVersion("/royalairways/
royalsw");

if ( existingVI.compareTo("3.2.1") <= 0 ) {
   // ... proceed to update ...
}
```

init

Initializes an `init` object.

Method of

`InstallVersion`

Syntax

```
init (
   int maj,
   int min,
   int rev,
   int bld);

init (
   String version);
```

Parameters

The `init` method has the following parameters:

`maj`	The major version number.
`min`	Minor version number.
`rev`	Revision number.
`bld`	Build number.
`version`	A string representing version information in the format "4.1.2.1234".

When maj, min, rev, and bld are provided as parameters, all four parameters are required, but all of them can be zero.

Chapter

4

File

Use the File object to manipulate local files and directories during the installation process.

File Overview

The File object has methods for analyzing the file system and preparing it (as when new directories, program shortcuts, version comparisons, or deletions are required) for newly installed software packages.

The File object works in conjunction with the *Install* object.

Method Reference

dirCreate	Creates a new directory.
dirGetParent	Returns an object representing the parent directory.
dirRemove	Removes a directory.
dirRename	Renames the specified directory.
copy	Makes a copy of the specified file.

diskSpaceAvailable	Returns the amount of disk space available in bytes.
exists	Returns a boolean value indicating whether the specified file exists or not.
execute	Queues a file for executing as part of the installation process.
exists	Returns a boolean value indicating whether an object represents a directory.
isFile	Returns a boolean value indicating whether an object is a file.
modDate	Returns a number representing the last modified date of the given file.
modDateChanged	Specifies whether the last modification on a file is older than a specified date
move	Moves a file from one location to another.
remove	Removes a file.
rename	Renames a file in place.
size	Returns the size of the file in bytes.
windowsShortcut	Creates a windows shortcut for a file.
macAlias	Creates a macintosh alias for a file.

dirCreate

Creates a new directory.

Method of

`File`

Syntax

```
int dirCreate( FileSpecObject dirToCreate );
```

Parameters

The `dirCreate` method has the following parameters:

`dirToCreate`	A FileSpecObject representing the pathname of the directory to create.

Returns

An integer error code. For a list of possible values, see *Return Codes.*

Description

The input parameter is a FileSpecObject that you have already created with the Install object's *getFolder* method. The following simple example demonstrates the use of the `dirCreate` method. Note that the `getFolder` method does not require that the folder or directory you specify exist in order for the object reference to be a valid one.

Example

```
f = getFolder("Program", "myNewDirectory");
err = File.dirCreate(f);
```

dirGetParent

Returns an object representing the parent directory of the current directory or file.

Method of

`File`

Syntax

```
FileSpecObject dirGetParent( FileSpecObject fileOrDir );
```

Parameters

The `dirGetParent` method has the following parameters:

`fileOrDir`	A FileSpecObject representing the pathname of the file or directory whose parent is being requested.

Returns

A FileSpecObject if successful; null if not successful.

Example

```
f = getFolder("Program", "myNewDirectory");
err = File.dirCreate(f);
err = File.getParent(f) // returns “Program”
```

dirRemove

Removes a directory.

Method of

`File`

Syntax

```
int dirRemove( FileSpecObject dirToRemove
    [, boolean recursive] );
```

Parameters

The `dirRemove` method has the following parameters:

`dirToRemove`	A FileSpecObject representing the directory to be removed.
recursive	An optional boolean value indicating whether the remove operation is to be performed recursively (1) or not (0).

Returns

An integer error code. For a list of possible values, see *Return Codes.*

dirRename

Renames a directory in place.

Method of

`File`

Syntax

```
int dirRename( FileSpecObject directory,
    String newname );
```

Parameters

The `dirRename` method has the following parameters:

`directory`	A FileSpecObject representing the directory to be renamed.
newname	The new name of the directory

Returns

An integer error code. For a list of possible values, see *Return Codes.*

copy

Makes a queued copy of the specified file.

Method of

`File`

Syntax

```
int copy( FileSpecObject source, FileSpecObject dest )
```

Parameters

The `copy` method has the following parameters:

`source`	A FileSpecObject object reprsenting the file to be copied.
`dest`	A FileSpecObject object representing the destination directory.

Returns

An integer error code. For a list of possible values, see *Return Codes.*

Description

The destination can be a directory or a filename. If destination does not exist a new file will be created.

diskSpaceAvailable

Returns the amount of disk space available in bytes on the local disk.

Method of

`File`

Syntax

```
double diskSpaceAvailable ( String NativeFolderPath );
```

Parameters

The `diskSpaceAvailable` method has the following parameters:

`NativeFolderPath`	A string representing the pathname of the partition, a file, or a directory on the partition whose space is being queried.

Returns

A double number representing the amount of space, in bytes, on the queried drive.

Description

Use this function to make sure there is adequate space on the local disk for extracting and installing your files (see example below). You can use a string representing any file on the disk you want to check, and XPInstall will resolve the reference to the partition itself.

Example

```
var diskAmtNeeded = 10000;
f = getFolder("Program");
diskSpace = File.diskSpaceAvailable(f);
g = getFolder(f, "myfile.txt");
if (diskSpace > diskAmtNeeded)
{
  err = addFile(..., ... g, ...);
  if (err == 0)
     performInstall();
  else
    cancelInstall();
```

```
}
else
{
  alert("not enough disk space. no install);
  cancelInstall();
}
```

execute

Queues the executing of a local file.

Method of

`File`

Syntax

```
int execute ( FileSpecObject executableFile,
    [String aParameters] );
```

Parameters

The `execute` method has the following parameters:

`executableFile`	A FileSpecObject representing the file to be executed.
`aParameters`	An optional parameter string that is passed to the executable. (Ignored on Mac OS)

Returns

An integer error code. For a list of possible values, see *Return Codes.*

Description

The specified file is not actually executed until the performInstall method is called. See *performInstall* for more information about queued commands during the installation process.

Example

```
f = getFolder("Program", "myTextEditor.exe");
err = File.execute(f, "myfile.txt");
// indicates that 'myfile.txt' will be
// opened in the editor
```

exists

Returns a value indicating whether the specified file or directory exists.

Method of

`File`

Syntax

```
boolean exists( FileSpecObject target )
```

Parameters

The `exists` method has the following parameters:

`target`	A FileSpecObject representing the file or directory being tested for existence.

Returns

A boolean value specifying whether the file or directory does indeed exist or does not.

Example

```
f = getFolder( “Program”, “sample.txt” );
if ( File.exists(f) ) // do something with the file
```

isDirectory

Returns a boolean value indicating whether the specified FileSpecObject is a directory.

Method of

`File`

Syntax

```
boolean isDirectory ( FileSpecObject NativeFolderPath );
```

Parameters

The `isDirectory` method has the following parameters:

`NativeFolderPath`	A FileSpecObject representing the queried directory.

Returns

A boolean value indicating whether the object is a directory or not.

isFile

Returns a boolean value indicating whether the given FileSpecObject is a file.

Method of

`File`

Syntax

```
boolean isFile (FileSpecObject NativeFolderPath);
```

Parameters

The `isFile` method has the following parameters:

`NativeFolderPath`	A FileSpecObject representing the queried file object.

Returns

A boolean value indicating whether the FileSpecObject is a file or not.

Example

```
f = getFolder( “Program”, “sample.txt” );
if ( File.isFile(f) ) // the object represents a file
```

modDate

Returns the last modified date of a specified file or directory.

Method of

`File`

Syntax

```
double modDate ( FileSpecObject NativeFolderPath );
```

Parameters

The `modDate` method has the following parameters:

`NativeFolderPath` A FileSpecObject representing the queried file.

Returns

A double number representing the date that the file was last modified.

Example

```
f = getFolder("Program");
fileSource = getFolder(f, "myfile.txt");
err = File.modDate(fileSource);
```

See *modDateChanged* for an example of comparing the dates of two files.

modDateChanged

Returns whether file has been modified since a certain date.

Method of

`File`

Syntax

```
boolean modDateChanged (FileSpecObject aSourceFolder,
    Number anOldDate);
```

Parameters

The `fileGetModDate` method has the following parameters:

`aSourceFolder`	A FileSpecObject representing the file to be queried.
`anOldDate`	A double representing the date.

Returns

A boolean value indicating whether the file has been modified since the input date or has not.

Description

Most often, the date passed in as the second parameter in modDateChanged is the returned value from a *modDate* on a separate file, as in the following example, in which the dates of two files are compared.

Example

```
fileSource1 = getFolder("Program", "file1.txt");
fileSource2 = getFolder("Program", "file2.txt");

err1 = File.modDate(fileSource1);    // the baseline returned
                                    //'time stamp' value

err2 = File.modDateChanged(fileSource1, err1);
logComment("File.modDateChanged should return 'false' = " +
err2);
// the reason it expects false is we're comparing
// the return 'time stamp' value for
// file1.txt with the actual file1.txt itself.
// Thus, no change in 'time stamp' values.

err3 = File.modDateChanged(fileSource2, err1);
logComment("File.modDateChanged should return 'true' = " +
err2);
// the reason it expects true is we're comparing
// the return 'time stamp' value  for
// file1.txt with another file, file2.txt, with a different
// 'time stamp' value.
```

move

Moves a file from one location to another.

Method of

`File`

Syntax

```
int move( FileSpecObject source, FileSpecObject dest);
```

Parameters

The `move` method has the following parameters:

`source`	A FileSpecObject representing the source file.
`dest`	A FileSpecObject representing the target directory.

Returns

An integer error code. For a list of possible values, see *Return Codes.*

Description

You must create a FileSpecObject for the destination directory to pass in for this function. If the destination doesn't already exist a file name for the destination is assumed.

Example

```
source = getFolder("Program", "file.txt");
dest = getFolder("Chrome");
err = File.move(source, dest);
```

remove

Deletes a specified file.

Method of

`File`

Syntax

```
int remove( FileSpecObject file )
```

Parameters

The `remove` method has the following parameters:

`file`	A FileSpecObject representing the file to be removed.

Returns

An integer error code. For a list of possible values, see *Return Codes.*

rename

Renames a specified file in place.

Method of

`File`

Syntax

```
int rename( FileSpecObject file, String newname )
```

Parameters

The `rename` method has the following parameters:

`file`	A FileSpecObject representing the file to be renamed.
newname	The new name of the file.

Returns

An integer error code. For a list of possible values, see *Return Codes.*

size

Return the size of the given file in bytes.

Method of

`File`

Syntax

```
int size (String NativeFolderPath);
```

Parameters

The `size` method has the following parameters:

`NativeFolderPath`	The full pathname to the file.

Returns

A number representing the size, in bytes, of the queried file.

windowsShortcut

Creates a windows shortcut to the installed software.

Method of

`File`

Syntax

```
int FileWindowsShortcut(
  FolderObject aTarget,
  FolderObject aShortcutPath,
  String aDescription,
  FolderObject aWorkingPath,
  String aParams,
  FolderObject aIcon,
  Number aIconId);
```

Parameters

The `windowsShortcut` method has the following parameters:

`aTarget`	A FileSpecObject representing the absolute path (including filename) to file that the shortcut will be created for.
`aShortcutPath`	A FileSpecObject representing the absolute path (not including filename) to where the shortcut is to be created.
`aDescription`	String description for the shortcut to be used as the shortcut name with a .lnk extension (do not include .lnk in the string).
`aWorkingPath`	A FileSpecObject representing the absolute path to the working dir for where aTarget will is to be run from.
`aParams`	Parameters that aTarget requires.
`aIcon`	A FileSpecObject representing the absolute path (including fiilename) to a file that contains icons. Can be either .ico, .dll, .exe, or any other binary file that contains icons.
`aIconId`	Index of the icon from aIcon to use for this shortcut.

Returns

An integer error code. For a list of possible values, see *Return Codes.*

Example

See *File.windowsShortcut* in the script examples chapter.

macAlias

Method of

`File`

Syntax

```
int macAlias(
   FileSpecObject destDir,
   String fileName,
   FileSpecObject aliasDir,
   String aliasName
);
```

Parameters

The `macAlias` method has the following parameters:

`destDir`	A FileSpecObject that represents the directory into which the program file will be installed.
`fileName`	A string representing the name of the file to be installed.
`aliasDir`	A FileSpecObject that represents the directory into which the alias file will be installed (e.g.,“Mac Desktop”).
`aliasName`	A string representing the name of the alias itself.

Returns

An integer error code. For a list of possible values, see *Return Codes.*

Example

`See` *File.macAlias* in the script examples chapter.

Chapter

5

WinProfile

(Windows only)

Windows developers use this object to manipulate the content of a Windows `.ini` file.

WinProfile Overview

This object does not have a public constructor. Instead, you construct an instance of this object by calling the `getWinProfile` method of the `Install` object. The two methods of the WinProfile object, *getString* and *writeString*, allow you to read and write the data in the key/value pairs of a Windows `.ini` file.

Method Reference

`getString`	Retrieves a value from a `.ini` file.
`writeString`	Changes a value in a `.ini` file.

getString

Retrieves a value from a `.ini` file.

Method of

`WinProfile`

Syntax

```
String getString (
   String section,
   String key);
```

Parameters

The method has the following parameters:

`section`	Section in the file, such as `"boot"` or `"drivers"`.
`key`	The key in that section whose value to return.

Returns

The value of the key or an empty string if none was found.

Description

The `getString` method is similar to the Windows API function `getPrivateProfileString`. Unlike that function, this method does not support using a null key to return a list of keys in a section.

Example

To get the name of the wallpaper file from the desktop section of the `win.ini` file, use this call:

```
ini = getWinProfile (getFolder("Windows"), "win.ini");
ini.getString ("Desktop", "Wallpaper");
```

writeString

Changes a value in a `.ini` file.

Method of

`WinProfile`

Syntax

```
Boolean writeString (
   String section,
   String key,
   String value);
```

Parameters

The method has the following parameters:

`section`	Section in the file, such as `"boot"` or `"drivers"`.
`key`	The key in that section whose value to change.
`value`	The new value.

Returns

True if successfully scheduled, otherwise, false.

Description

The `writeString` method is similar to the Windows API function `WritePrivateProfileString`. To delete an existing value, supply null as the `value` parameter. Unlike the `WritePrivateProfileString` function, this method does not support using a null key to delete an entire section.

Values are not actually written until `performInstall` is called.

Example

To set the name of the wallpaper file from the desktop section of the `win.ini` file, use this call:

```
ini = getWinProfile (getFolder("Windows"), "win.ini");
ini.writeString ("Desktop", "Wallpaper", "newpathname");
```

Chapter

6

WinReg

(Windows only)

Windows developers use this object to manipulate the content of the Windows registry.

WinReg Overview

This object does not have a public constructor. Instead, you construct an instance of this object by calling the `getWinRegistry` method of the `Install` object.

This discussion assumes you are already familiar with the Windows Registry. For information on it, see API documentation for Windows NT or Windows 95.

When you construct a `WinReg` object, it is set to operate with `HKEY_CLASSES_ROOT` as its root key. To use a different root key, use the `setRootKey` method. Typically values in the Windows Registry are strings. To manipulate such values, use the `getValueString` and `setValueString` methods. To manipulate other values, use the `getValue` and `setValue` methods.

Reading registry values is immediate. However, writing to the registry is delayed until `performInstall` is called.

Method Summary

createKey	Adds a key.
deleteKey	Removes a key.
deleteValue	Removes the value of an arbitrary key.
getValue	Retrieves the value of an arbitrary key.
getValueNumber	Retrieves the value of a key, when that value is an integer.
getValueString	Retrieves the value of a key, when that value is a string.
setRootKey	Changes the root key being accessed.
setValue	Sets the value of an arbitrary key.
setValueNumber	Sets the value of a key, when that value is an integer.
setValueString	Sets the value of a key, when that value is a string.
WinRegValue	Creates a `WinRegValue` object.

createKey

Adds a key to the registry.

Method of

`WinReg`

Syntax

```
int createKey (
   String subkey,
   String classname);
```

Parameters

The method has the following parameters:

`subkey`	The key path to the appropriate location in the key hierarchy, such as `"Software\\Netscape\\Navigator\\Mail"`.
`classname`	Usually an empty string. For information on this parameter, see the description of `RegCreateKeyEx` in your Windows API documentation.

Returns

0 if it succeeded; a nonzero number if it failed to schedule the creation. For a list of possible values, see *Return Codes*.

Description

The `createKey` method adds a key to the registry. You must add a key to the registry before you can add a value for that key.

deleteKey

Removes a key from the registry.

Method of

`WinReg`

Syntax

```
int deleteKey (
   String subkey);
```

Parameters

The method has the following parameters:

`subkey`	The key path to the appropriate location in the key hierarchy, such as `"Software\\Netscape\\Navigator\\Mail"`.

Returns

0 if it succeeded; a nonzero number if it failed to schedule the deletion. For a list of possible values, see *Return Codes*.

deleteValue

Removes the value of an arbitrary key.

Method of

`WinReg`

Syntax

```
int deleteValue (
   String subkey,
   String valname);
```

Parameters

The `deleteValue` method has the following parameters:

`subkey`	The key path to the appropriate location in the key hierarchy, such as `"Software\\Netscape\\Navigator\\Mail"`.
`valname`	The name of the value-name/value pair you want to remove.

Returns

0 if it succeeded; a nonzero number if it failed to schedule the deletion.

getValue

Netscape 6 and Mozilla do not currently support this method.

Retrieves the value of an arbitrary key.

Method of

`WinReg`

Syntax

```
WinRegValue getValue (
   String subkey,
   String valname);
```

Parameters

The `getValue` method has the following parameters:

`subkey`	The key path to the appropriate location in the key hierarchy, such as `"Software\\Netscape\\Navigator\\Mail"`.
`valname`	The name of the value-name/value pair whose value you want.

Returns

A `WinRegValue` object representing the value of the named value-name/value pair or null if there is no value or if there is an error. See `WinRegValue` for information about these values.

Description

The `getValue` method retrieves the value of an arbitraty key. Use this method if the value you want is not a string. If the value is a string, the `getValueString` method is more convenient.

getValueNumber

Gets the value of a key when that value is a number.

Method of

`File`

Syntax

```
Number getValueNumber (
   String subkey,
   String valname);
```

Parameters

The `getValueString` method has the following parameters:

`subkey`	The key path to the appropriate location in the key hierarchy, such as `"Software\\Netscape\\Navigator\\Mail"`.
`valname`	The name of the value-name/value pair whose value you want.

Returns

Number value of the specified key or null if there's an error, the value is not found, or the value is not a string.

getValueString

Retrieves the value of a key when that value is a string.

Method of

`WinReg`

Syntax

```
String getValueString (
   String subkey,
   String valname);
```

Parameters

The `getValueString` method has the following parameters:

`subkey`	The key path to the appropriate location in the key hierarchy, such as `"Software\\Netscape\\Navigator\\Mail"`.
`valname`	The name of the value-name/value pair whose value you want.

Returns

A string representing the value of the named value-name/value pair or null if there's an error, the value is not found, or the value is not a string.

Description

The `getValueString` method gets the value of a string. If the value is not a string, use the `getValue` method instead.

setRootKey

Changes the root key being accessed.

Method of

`WinReg`

Syntax

```
void setRootKey (
   int key);
```

Parameters

The method has the following parameter:

`key`	An integer representing a root key in the registry.

Returns

Nothing.

Description

The `setRootKey` changes the root key. When you create a `WinReg` object, it is set to access keys under the `HKEY_CLASSES_ROOT` portion of the registry. If you want to access keys in another portion, you must use this method to change the root key.

On 16-bit Windows platforms, `HKEY_CLASSES_ROOT` is the only valid value and this method does nothing.

These root keys are represented as fields of the `WinReg` object. The values you can use are:

- `HKEY_CLASSES_ROOT`
- `HKEY_CURRENT_USER`

- `HKEY_LOCAL_MACHINE`
- `HKEY_USERS`

Example

To use the `HKEY_USERS` section, use these statements:

```
winreg = getWinRegistry();
winreg.setRootKey(winreg.HKEY_USERS);
```

setValue

Netscape 6 and Mozilla do not currently support this method.

Sets the value of an arbitrary key.

Method of

`WinReg`

Syntax

```
String setValue (
   String subkey,
   String valname,
   WinRegValue value);
```

Parameters

The `setValue` method has the following parameters:

`subkey`	The key path to the appropriate location in the key hierarchy, such as `"Software\\Netscape\\Navigator\\Mail"`.
`valname`	The name of the value-name/value pair whose value you want to change.
`value`	A `WinRegValue` object representing the new non-string value. See `WinRegValue` for information about these values.

Returns

0 if it succeeded; a nonzero number if it failed to schedule the action. For a list of possible values, see *Return Codes*.

Description

The `setValue` method sets the value of an arbitrary key. Use this method if the value you want to set is not a string. If the value is a string, the `setValueString` method is more convenient.

setValueNumber

Sets the value of a key, when that value is a number.

Method of

`WinReg`

Syntax

```
int setValueString (
   String subkey,
   String valname,
   Number value);
```

Parameters

The method has the following parameters:

`subkey`	The key path to the appropriate location in the key hierarchy, such as `"Software\\Netscape\\Navigator\\Mail"`.
`valname`	The name of the value-name/value pair whose value you want to change.
`value`	A number representing the new string value.

Returns

0 if it succeeded; a nonzero number if it failed to schedule the action. For a list of possible values, see *Return Codes*.

Description

The `setValueNumber` method sets the value of a key when that value is a number. Use this method if the value you want to set is a number. If the value is not a string, use the `setValue` or `setValueString` methods instead.

setValueString

Sets the value of a key, when that value is a string.

Method of

`WinReg`

Syntax

```
int setValueString (
   String subkey,
   String valname,
   String value);
```

Parameters

The method has the following parameters:

`subkey`	The key path to the appropriate location in the key hierarchy, such as `"Software\\Netscape\\Navigator\\Mail"`.
`valname`	The name of the value-name/value pair whose value you want to change.
`value`	The new string value.

Returns

0 if it succeeded; a nonzero number if it failed to schedule the action. For a list of possible values, see *Return Codes*.

Description

The `setValueString` method sets the value of a key when that value is a string. Use this method if the value you want to set is a string. If the value is not a string, use the `setValue` method instead.

WinRegValue

(Windows only)

Creates a `WinRegValue` object.

Syntax

```
WinRegValue (
   int datatype,
   byte[] regdata);
```

Parameters

The `WinRegValue` constructor takes the following parameter:

`datatype` An integer indicating the type of the data encapsulated by this object. The possible values are:

- `WinRegValue.REG_SZ = 1`
- `WinRegValue.REG_EXPAND_SZ = 2`
- `WinRegValue.REG_BINARY = 3`
- `WinRegValue.REG_DWORD = 4`
- `WinRegValue.REG_DWORD_LITTLE_ENDIAN = 4`
- `WinRegValue.REG_DWORD_BIG_ENDIAN = 5`
- `WinRegValue.REG_LINK = 6`
- `WinRegValue.REG_MULTI_SZ = 7`
- `WinRegValue.REG_RESOURCE_LIST = 8`
- `WinRegValue.REG_FULL_RESOURCE_DESCRIPTOR = 9`
- `WinRegValue.REG_RESOURCE_REQUIREMENTS_LIST = 10`

`regdata` A Java byte array containing the data.

Returns

A new `WinRegValue` object, with the data members `type` and `data` set to the values passed to this constructor.

Description

Advanced Windows developers can use this object to manipulate non-string values for the Windows Registry. An object of this type has two fields: the type of the data and the value. For information on the possible data types for a registry value, see your Windows API documentation. You supply the value for these fields to the constructor for this class.

7

Return Codes

The methods described in this chapter can return any of the following return codes. In Communicator 4.5, these constants are defined as part of the SoftwareUpdate object.

Name	Code	Explanation
SUCCESS	0	Success.
REBOOT_NEEDED	999	The files were installed, but one or more components were in use. Restart the computer and Communicator to complete the installation process. On Windows NT, you may only need to restart Communicator as long as you did not replace operating system files.
BAD_PACKAGE_NAME	-200	A problem occurred with the package name supplied to *initInstall*
UNEXPECTED_ERROR	-201	An unrecognized error occurred.
ACCESS_DENIED	-202	The user did not grant the required security privilege.
TOO_MANY_CERTIFICATES	-203	Installation script was signed by more than one certificate
NO_INSTALL_SCRIPT	-204	Installation script was not signed
NO_CERTIFICATE	-205	Extracted file is not signed or the file (and, therefore, its certificate) could not be found.

NO_MATCHING_CERTIFICATE	-206	Extracted file was not signed by the certificate used to sign the installation script
CANT_READ_ARCHIVE	-207	XPI package cannot be read
INVALID_ARGUMENTS	-208	Bad parameters to a function
ILLEGAL_RELATIVE_PATH	-209	Illegal relative path
USER_CANCELLED	-210	User clicked Cancel on Install dialog
INSTALL_NOT_STARTED	-211	A problem occurred with the parameters to *initInstall*, or *initInstall* was not called first
SILENT_MODE_DENIED	-212	The silent installation privilege has not been granted.
NO_SUCH_COMPONENT	-213	The specified component is not present in the Client Version Registry.
DOES_NOT_EXIST	-214	The specified file cannot be deleted because it does not exist.
READ_ONLY	-215	The specified file cannot be deleted because its permissions are set to read only.
IS_DIRECTORY	-216	The specified file cannot be deleted because it is a directory.
NETWORK_FILE_IS_IN_USE	-217	The specified file cannot be deleted because it is in use.
APPLE_SINGLE_ERR	-218	An error occurred when unpacking a file in AppleSingle format.
INVALID_PATH_ERR	-219	The path provided to *getFolder* was invalid.
PATCH_BAD_DIFF	-220	An error occurred in GDIFF.
PATCH_BAD_CHECKSUM_TARGET	-221	The checksum generated for the source file does not match the checksum in the XPI file.
PATCH_BAD_CHECKSUM_RESULT	-222	The checksum of the patched file failed.
UNINSTALL_FAILED	-223	An error occurred while uninstalling a package.
PACKAGE_FOLDER_NOT_SET	-224	Install folder not set in installation script
EXTRACTION_FAILED	-225	Extraction of XPI file failed.

FILENAME_ALREADY_USED	-226	Same filename being used in install
INSTALL_CANCELLED	-227	Raised when installation is cancelled in medias res.
DOWNLOAD_ERROR	-228	Problem with download
SCRIPT_ERROR	-229	Error in the script
ALREADY_EXISTS	-230	File already exists locally
IS_FILE	-231	Expected targer directoy and got file
SOURCE_DOES_NOT_EXIST	-232	Source file/dir not found
SOURCE_IS_DIRECTORY	-233	Expected file and got directory
SOURCE_IS_FILE	-234	Expected directory and got file
INSUFFICIENT_DISK_SPACE	-235	Not enough disk space for install
FILENAME_TOO_LONG	-236	
UNABLE_TO_LOCATE_LIB_FUNCTION	-237	
UNABLE_TO_LOAD_LIBRARY	-238	
CHROME_REGISTRY_ERROR	-239	
MALFORMED_INSTALL	-240	
OUT_OF_MEMORY	-299	Insufficient memory for operation
GESTALT_UNKNOWN_ERROR	-5550	
GESTALT_INVALID_ARGUMENT	-5551	

8

Script Examples

The following short list of examples demonstrates some of the principal installation functions in the XPInstall API:

- *InstallTrigger.installChrome*
- *InstallTrigger.startSoftwareUpdate*
- *[Install.]addFile*
- *[Install.]addDirectory*
- *File.windowsShortcut*
- *File.macAlias*
- *Windows Install Example*

Trigger Scripts and Install Scripts

Trigger scripts are simple installations that can be initiated from event handlers and other JavaScript code on a web page. Triggers use the *InstallTrigger* object methods to compare software versions, install chrome, and perform simple software installations.

Install scripts use the *Install*, *File*, *InstallVersion* and platform-specific installation object methods to initialize, queue, manage, and perform the installation of one or more software packages. These install scripts are typically located at the top level of the XPI archives in which the installations are stored. A trigger script may trigger the downloading of a XPI, which in turn will use its own install.js script to manage the complete installation.

InstallTrigger.installChrome

Trigger scripts are typically invoked by JavaScript event handlers on hyperlinks. When a user clicks the link “Install the New Blue theme” in the example below, XPInstall downloads, registers, and installs the theme contained in newblue.xpi to the user’s profile directory.

```
<a href="#"
  onclick="InstallTrigger.installChrome(
     InstallTrigger.SKIN,
     'http://wildskins/newblue.xpi',
     'newblue/1.0');">
Install the New Blue theme</a>
```

InstallTrigger.startSoftwareUpdate

This is a very simple example of the InstallTrigger object’s principal method, *startSoftwareUpdate*, which takes a string representing the path to the XPI and installs that XPI on the local machine.

```
function triggerURL(url)
{
   InstallTrigger.startSoftwareUpdtate(url);
}
// get the url to the .xpi from either a form
// or text field entry. Then do:
... onclick="triggerURL(this.form.url.value);
```

[Install.]addFile

The Install object’s *addFile* method is the standard way to queue files for installation.

```
var xpiSrc = "file.txt";
initInstall(
  "Adding A File",
  "testFile",
  "1.0.1.7",
  1);
```

```
f = getFolder("Program");
setPackageFolder(f);
addFile(xpiSrc);
if (0 == getLastError())
   performInstall();
else
   cancelInstall();
```

[Install.]addDirectory

The Install object's *addDirectory* method queues an entire directory for installation once *performInstall* is called.

```
var vi = "10.10.10.10";
var xpiSrc = "adddir1";

initInstall("addFileNoVers1", "adddir_1", vi, 1);

f = getFolder("Program");
setPackageFolder(f);
err = addDirectory(xpiSrc);
logComment("the error = " + err);

if (0 == getLastError())
   performInstall();
else
   cancelInstall();
```

File.windowsShortcut

In this example, the *windowsShortcut* method is used to add a shortcut in the Program directory ("Program" is a keyword for the directory in which the program itself is installed, for example, C:\Program Files\Netscape\Netscape 6\" on Windows) to Windows software (misc.exe) that is installed in the "Windows" directory.

```
var xpiSrc = "misc.exe";
var vi = "1.1.1.1";
initInstall(
  "Windows Shortcut",
```

```
  "test",
  vi,
  0);
f = getFolder("Windows");
g = getFolder("Temporary");
addFile(
  "miscshortcut",
  "2.2.2.2",
  xpiSrc,
  f,
  xpiSrc,
  true);
target = getFolder(f, xpiSrc);
shortcutpath = getFolder("Program");
err = File.windowsShortcut(
  target,
  shortcutpath,
  "misc shortcut",
  g,
  "",
  target,
  0);
logComment("file.windowsShortcut returns: " + err);
if (0 == getLastError())
  performInstall();
else
  cancelInstall();
```

File.macAlias

In this example, a mac alias is created for software (AppleCD Audio Player) that is installed locally.

```
xpiSrc = "Miscellaneous Program";
var vi = "1.1.1.1";
initInstall(
  "Macintosh Alias",
  "misc",
  vi,
  0);
f = getFolder("Program");
g = getFolder("Mac Desktop");
```

```
addFile(
  "filemacalias",
  "2.2.2.2",
  xpiSrc,
  f,
  xpiSrc,
  true);
err = File.macAlias(f, xpiSrc, g, xpiSrc + " alias");
logComment("File.macAlias returns: " + err);
if (0 == getLastError())
   performInstall();
else
   cancelInstall();
```

Windows Install Example

This example shows the installation of a XPI in which user profile information is contained. Note the disk space verification, the editing of the Windows registry, the writing to the installation log, and the error checking before either *performInstall* or *cancelInstall* is called.

```
// this function verifies disk space in kilobytes
function verifyDiskSpace(dirPath, spaceRequired)
{
  var spaceAvailable;

  // Get the available disk space on the given path
  spaceAvailable = fileGetDiskSpaceAvailable(dirPath);

  // Convert the available disk space into kilobytes
  spaceAvailable = parseInt(spaceAvailable / 1024);

  // do the verification
  if(spaceAvailable < spaceRequired)
  {
    logComment("Insufficient disk space: " + dirPath);
    logComment("  required : " + spaceRequired + " K");
    logComment("  available: " + spaceAvailable + " K");
    return(false);
  }

  return(true);
}
```

```
function updateWinReg4Ren8dot3()
{
  var fProgram      = getFolder("Program");
  var fTemp         = getFolder("Temporary");

  //Notes:
  // can't use a double backslash before subkey
  // - Windows already puts it in.
  // subkeys have to exist before values can be put in.
  var subkey;  // the name of the subkey you are poking around in
  var valname; // the name of the value you want to look at
  var value;   // the data in the value you want to look at.
  var winreg = getWinRegistry() ;

  if(winreg != null)
  {
    // Here, we get the current version.
    winreg.setRootKey(winreg.HKEY_CURRENT_USER) ;// CURRENT_USER
    subkey  =
      "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce" ;

    winreg.createKey(subkey,"");
    valname = "ren8dot3";
    value = fProgram + "ren8dot3.exe " + fTemp + "ren8dot3.ini";
    err     = winreg.setValueString(subkey, valname, value);
  }
}

function prepareRen8dot3(listLongFilePaths)
{
  var fTemp        = getFolder("Temporary");
  var fProgram     = getFolder("Program");
  var fRen8dot3Ini = getWinProfile(fTemp, "ren8dot3.ini");
  var bIniCreated  = false;
  var fLongFilePath;
  var sShortFilePath;

  if(fRen8dot3Ini != null)
  {
    for(i = 0; i < listLongFilePaths.length; i++)
    {
      fLongFilePath = getFolder(fProgram, listLongFilePaths[i]);
      sShortFilePath  = File.windowsGetShortName(fLongFilePath);
      if(sShortFilePath)
      {
```

```
        fRen8dot3Ini.writeString("rename",
                sShortFilePath, fLongFilePath);
        bIniCreated = true;
      }
    }

    if(bIniCreated)
      updateWinReg4Ren8dot3() ;
  }

  return(0);
}

// main
var srDest;
var err;
var fProgram;

srDest = 449;
err    = initInstall(prettyName, regName, "6.0.0.2000110801");
logComment("initInstall: " + err);

fProgram = getFolder("Program");
logComment("fProgram: " + fProgram);

if(verifyDiskSpace(fProgram, srDest))
{
  setPackageFolder(fProgram);
  err = addDirectory("",
    "6.0.0.2000110801",
    "bin", // dir name in jar to extract
    fProgram, // Where to put this file
                // (Returned from GetFolder)
    "", // subdir name to create relative to fProgram
    true); // Force Flag
  logComment("addDirectory() returned: " + err);

  // check return value
  if(err == SUCCESS)
  {
    err = performInstall();
    logComment("performInstall() returned: " + err);
  }
  else
    cancelInstall(err);
}
```

```
else
  cancelInstall(INSUFFICIENT_DISK_SPACE);

// end main
```