35.1. `msilib` — Read and write Microsoft Installer files¶

Platforms: Windows

New in version 2.5.

The `msilib` supports the creation of Microsoft Installer (`.msi`) files. Because these files often contain an embedded "cabinet" file (`.cab`), it also exposes an API to create CAB files. Support for reading `.cab` files is currently not implemented; read support for the `.msi` database is possible.

This package aims to provide complete access to all tables in an `.msi` file, therefore, it is a fairly low-level API. Two primary applications of this package are the `distutils` command `bdist_msi`, and the creation of Python installer package itself (although that currently uses a different version of `msilib`).

The package contents can be roughly split into four parts: low-level CAB routines, low-level MSI routines, higher-level MSI routines, and standard table structures.

`msilib.FCICreate`(*cabname*, *files*)¶

Create a new CAB file named *cabname*. *files* must be a list of tuples, each containing the name of the file on disk, and the name of the file inside the CAB file.

The files are added to the CAB file in the order they appear in the list. All files are added into a single CAB file, using the MSZIP compression algorithm.

Callbacks to Python for the various steps of MSI creation are currently not exposed.

`msilib.UuidCreate`()¶
Return the string representation of a new unique identifier. This wraps the Windows API functions `UuidCreate()` and `UuidToString()`.

`msilib.OpenDatabase`(*path*, *persist*)¶
Return a new database object by calling MsiOpenDatabase. *path* is the file name of the MSI file; *persist* can be one of the constants `MSIDBOPEN_CREATEDIRECT`, `MSIDBOPEN_CREATE`, `MSIDBOPEN_DIRECT`, `MSIDBOPEN_READONLY`, or `MSIDBOPEN_TRANSACT`, and may include the flag `MSIDBOPEN_PATCHFILE`. See the Microsoft documentation for the meaning of these flags; depending on the flags, an existing database is opened, or a new one created.

`msilib.CreateRecord`(*count*)¶
Return a new record object by calling `MSICreateRecord()`. *count* is the number of fields of the record.

`msilib.init_database`(*name*, *schema*, *ProductName*, *ProductCode*, *ProductVersion*, *Manufacturer*)¶

Create and return a new database *name*, initialize it with *schema*, and set the properties *ProductName*, *ProductCode*, *ProductVersion*, and *Manufacturer*.

schema must be a module object containing `tables` and `_Validation_records` attributes; typically, `msilib.schema` should be used.

The database will contain just the schema and the validation records when this function returns.

`msilib.add_data`(*database*, *table*, *records*)¶

Add all *records* to the table named *table* in *database*.

The *table* argument must be one of the predefined tables in the MSI schema, e.g. `'Feature'`, `'File'`, `'Component'`, `'Dialog'`, `'Control'`, etc.

records should be a list of tuples, each one containing all fields of a record according to the schema of the table. For optional fields, `None` can be passed.

Field values can be int or long numbers, strings, or instances of the Binary class.

class `msilib.Binary`(*filename*)¶
Represents entries in the Binary table; inserting such an object using `add_data()` reads the file named *filename* into the table.

`msilib.add_tables`(*database*, *module*)¶

Add all table content from *module* to *database*. *module* must contain an attribute *tables* listing all tables for which content should be added, and one attribute per table that has the actual content.

This is typically used to install the sequence tables.

`msilib.add_stream`(*database*, *name*, *path*)¶

Add the file *path* into the `_Stream` table of *database*, with the stream name *name*.

`msilib.gen_uuid`()¶
Return a new UUID, in the format that MSI typically requires (i.e. in curly braces, and with all hexdigits in upper-case).

See also

FCICreateFile UuidCreate UuidToString

35.1.1. Database Objects¶

`Database.OpenView`(*sql*)¶
Return a view object, by calling `MSIDatabaseOpenView()`. *sql* is the SQL statement to execute.

`Database.Commit`()¶
Commit the changes pending in the current transaction, by calling `MSIDatabaseCommit()`.

`Database.GetSummaryInformation`(*count*)¶
Return a new summary information object, by calling `MsiGetSummaryInformation()`. *count* is the maximum number of updated values.

See also

MSIDatabaseOpenView MSIDatabaseCommit MSIGetSummaryInformation

35.1.2. View Objects¶

`View.Execute`(*params*)¶
Execute the SQL query of the view, through `MSIViewExecute()`. If *params* is not `None`, it is a record describing actual values of the parameter tokens in the query.

`View.GetColumnInfo`(*kind*)¶
Return a record describing the columns of the view, through calling `MsiViewGetColumnInfo()`. *kind* can be either `MSICOLINFO_NAMES` or `MSICOLINFO_TYPES`.

`View.Fetch`()¶
Return a result record of the query, through calling `MsiViewFetch()`.

`View.Modify`(*kind*, *data*)¶

Modify the view, by calling `MsiViewModify()`. *kind* can be one of `MSIMODIFY_SEEK`, `MSIMODIFY_REFRESH`, `MSIMODIFY_INSERT`, `MSIMODIFY_UPDATE`, `MSIMODIFY_ASSIGN`, `MSIMODIFY_REPLACE`, `MSIMODIFY_MERGE`, `MSIMODIFY_DELETE`, `MSIMODIFY_INSERT_TEMPORARY`, `MSIMODIFY_VALIDATE`, `MSIMODIFY_VALIDATE_NEW`, `MSIMODIFY_VALIDATE_FIELD`, or `MSIMODIFY_VALIDATE_DELETE`.

data must be a record describing the new data.

`View.Close`()¶
Close the view, through `MsiViewClose()`.

See also

MsiViewExecute MSIViewGetColumnInfo MsiViewFetch MsiViewModify MsiViewClose

35.1.3. Summary Information Objects¶

`SummaryInformation.GetProperty`(*field*)¶
Return a property of the summary, through `MsiSummaryInfoGetProperty()`. *field* is the name of the property, and can be one of the constants `PID_CODEPAGE`, `PID_TITLE`, `PID_SUBJECT`, `PID_AUTHOR`, `PID_KEYWORDS`, `PID_COMMENTS`, `PID_TEMPLATE`, `PID_LASTAUTHOR`, `PID_REVNUMBER`, `PID_LASTPRINTED`, `PID_CREATE_DTM`, `PID_LASTSAVE_DTM`, `PID_PAGECOUNT`, `PID_WORDCOUNT`, `PID_CHARCOUNT`, `PID_APPNAME`, or `PID_SECURITY`.

`SummaryInformation.GetPropertyCount`()¶
Return the number of summary properties, through `MsiSummaryInfoGetPropertyCount()`.

`SummaryInformation.SetProperty`(*field*, *value*)¶
Set a property through `MsiSummaryInfoSetProperty()`. *field* can have the same values as in `GetProperty()`, *value* is the new value of the property. Possible value types are integer and string.

`SummaryInformation.Persist`()¶
Write the modified properties to the summary information stream, using `MsiSummaryInfoPersist()`.

See also

MsiSummaryInfoGetProperty MsiSummaryInfoGetPropertyCount MsiSummaryInfoSetProperty MsiSummaryInfoPersist

35.1.4. Record Objects¶

`Record.GetFieldCount`()¶
Return the number of fields of the record, through `MsiRecordGetFieldCount()`.

`Record.GetInteger`(*field*)¶
Return the value of *field* as an integer where possible. *field* must be an integer.

`Record.GetString`(*field*)¶
Return the value of *field* as a string where possible. *field* must be an integer.

`Record.SetString`(*field*, *value*)¶
Set *field* to *value* through `MsiRecordSetString()`. *field* must be an integer; *value* a string.

`Record.SetStream`(*field*, *value*)¶
Set *field* to the contents of the file named *value*, through `MsiRecordSetStream()`. *field* must be an integer; *value* a string.

`Record.SetInteger`(*field*, *value*)¶
Set *field* to *value* through `MsiRecordSetInteger()`. Both *field* and *value* must be an integer.

`Record.ClearData`()¶
Set all fields of the record to 0, through `MsiRecordClearData()`.

See also

MsiRecordGetFieldCount MsiRecordSetString MsiRecordSetStream MsiRecordSetInteger MsiRecordClear

35.1.5. Errors¶

All wrappers around MSI functions raise `MsiError`; the string inside the exception will contain more detail.

35.1.6. CAB Objects¶

class `msilib.CAB`(*name*)¶

The class CAB represents a CAB file. During MSI construction, files will be added simultaneously to the `Files` table, and to a CAB file. Then, when all files have been added, the CAB file can be written, then added to the MSI file.

name is the name of the CAB file in the MSI file.

`append`(*full*, *file*, *logical*)¶

Add the file with the pathname *full* to the CAB file, under the name *logical*. If there is already a file named *logical*, a new file name is created.

Return the index of the file in the CAB file, and the new name of the file inside the CAB file.

`commit`(*database*)¶
Generate a CAB file, add it as a stream to the MSI file, put it into the `Media` table, and remove the generated file from the disk.

35.1.7. Directory Objects¶

class `msilib.Directory`(*database*, *cab*, *basedir*, *physical*, *logical*, *default*, *component*[, *componentflags*])¶

Create a new directory in the Directory table. There is a current component at each point in time for the directory, which is either explicitly created through `start_component()`, or implicitly when files are added for the first time. Files are added into the current component, and into the cab file. To create a directory, a base directory object needs to be specified (can be `None`), the path to the physical directory, and a logical directory name. *default* specifies the DefaultDir slot in the directory table. *componentflags* specifies the default flags that new components get.

`start_component`([*component*[, *feature*[, *flags*[, *keyfile*[, *uuid*]]]]])¶
Add an entry to the Component table, and make this component the current component for this directory. If no component name is given, the directory name is used. If no *feature* is given, the current feature is used. If no *flags* are given, the directory's default flags are used. If no *keyfile* is given, the KeyPath is left null in the Component table.

`add_file`(*file*[, *src*[, *version*[, *language*]]])¶
Add a file to the current component of the directory, starting a new one if there is no current component. By default, the file name in the source and the file table will be identical. If the *src* file is specified, it is interpreted relative to the current directory. Optionally, a *version* and a *language* can be specified for the entry in the File table.

`glob`(*pattern*[, *exclude*])¶
Add a list of files to the current component as specified in the glob pattern. Individual files can be excluded in the *exclude* list.

`remove_pyc`()¶
Remove `.pyc/.pyo` files on uninstall.

See also

Directory Table File Table Component Table FeatureComponents Table

35.1.8. Features¶

class `msilib.Feature`(*database*, *id*, *title*, *desc*, *display*[, *level=1*[, *parent*[, *directory*[, *attributes=0*]]]])¶

Add a new record to the `Feature` table, using the values *id*, *parent.id*, *title*, *desc*, *display*, *level*, *directory*, and *attributes*. The resulting feature object can be passed to the `start_component()` method of `Directory`.

`set_current`()¶
Make this feature the current feature of `msilib`. New components are automatically added to the default feature, unless a feature is explicitly specified.

See also

Feature Table

35.1.9. GUI classes¶

`msilib` provides several classes that wrap the GUI tables in an MSI database. However, no standard user interface is provided; use `bdist_msi` to create MSI files with a user-interface for installing Python packages.

class `msilib.Control`(*dlg*, *name*)¶

Base class of the dialog controls. *dlg* is the dialog object the control belongs to, and *name* is the control's name.

`event`(*event*, *argument*[, *condition=1*[, *ordering*]])¶
Make an entry into the `ControlEvent` table for this control.

`mapping`(*event*, *attribute*)¶
Make an entry into the `EventMapping` table for this control.

`condition`(*action*, *condition*)¶
Make an entry into the `ControlCondition` table for this control.

class `msilib.RadioButtonGroup`(*dlg*, *name*, *property*)¶

Create a radio button control named *name*. *property* is the installer property that gets set when a radio button is selected.

`add`(*name*, *x*, *y*, *width*, *height*, *text*[, *value*])¶
Add a radio button named *name* to the group, at the coordinates *x*, *y*, *width*, *height*, and with the label *text*. If *value* is omitted, it defaults to *name*.

class `msilib.Dialog`(*db*, *name*, *x*, *y*, *w*, *h*, *attr*, *title*, *first*, *default*, *cancel*)¶

Return a new `Dialog` object. An entry in the `Dialog` table is made, with the specified coordinates, dialog attributes, title, name of the first, default, and cancel controls.

`control`(*name*, *type*, *x*, *y*, *width*, *height*, *attributes*, *property*, *text*, *control_next*, *help*)¶

Return a new `Control` object. An entry in the `Control` table is made with the specified parameters.

This is a generic method; for specific types, specialized methods are provided.

`text`(*name*, *x*, *y*, *width*, *height*, *attributes*, *text*)¶
Add and return a `Text` control.

`bitmap`(*name*, *x*, *y*, *width*, *height*, *text*)¶
Add and return a `Bitmap` control.

`line`(*name*, *x*, *y*, *width*, *height*)¶
Add and return a `Line` control.

`pushbutton`(*name*, *x*, *y*, *width*, *height*, *attributes*, *text*, *next_control*)¶
Add and return a `PushButton` control.

`radiogroup`(*name*, *x*, *y*, *width*, *height*, *attributes*, *property*, *text*, *next_control*)¶
Add and return a `RadioButtonGroup` control.

`checkbox`(*name*, *x*, *y*, *width*, *height*, *attributes*, *property*, *text*, *next_control*)¶
Add and return a `CheckBox` control.

See also

Dialog Table Control Table Control Types ControlCondition Table ControlEvent Table EventMapping Table RadioButton Table

35.1.10. Precomputed tables¶

`msilib` provides a few subpackages that contain only schema and table definitions. Currently, these definitions are based on MSI version 2.0.

`msilib.schema`¶

This is the standard MSI schema for MSI 2.0, with the *tables* variable providing a list of table definitions, and *_Validation_records* providing the data for MSI validation.

`msilib.sequence`¶

This module contains table contents for the standard sequence tables: *AdminExecuteSequence*, *AdminUISequence*, *AdvtExecuteSequence*, *InstallExecuteSequence*, and *InstallUISequence*.

`msilib.text`¶

This module contains definitions for the UIText and ActionText tables, for the standard installer actions.

Table Of Contents

Previous topic

35. MS Windows Specific Services

Next topic

35.2. `msvcrt` – Useful routines from the MS VC++ runtime

This Page

- Show Source

Navigation