pyTSon

API Documentation

May 10, 2017

Contents

1 Module devtools

1.1 Functions

installedPackages()

Returns a list of installed packages (installed with pip).

Return Value

a list of dictionaries containing name, version, directory (dir) and dist-info directory (distdir)

(type=list[dict{str: str}])

removePackage(*name*, *version*)

Removes a package (installed with pip). Throws an exception if the package could not be found

Parameters

`name:` the name of the package

(type=str)

`version:` the version string of the package

(type=str)

1.2 Class PluginInstaller

object
└ **devtools.PluginInstaller**

Class used to install new python plugins and its dependencies.

1.2.1 Methods

__init__(*self*, *stdout*=`None`)

x.__init__(...) initializes x; see help(type(x)) for signature

Parameters

`stdout:` A callable used as print function (takes str argument); defaults to None; if None stdout is used instead

(type=callable)

Overrides: object.__init__

createPlugin(*name*, *withfile*=`True`, *content*=`None`)

Creates the infrastructure for a new plugin.

Parameters

`name:` the name of the plugin

(type=str)

`withfile:` if True, the file __init__.py is created in the plugin directory, defaults to True

(type=bool)

`content:` content of __ini__.py; defaults to None; if None, an empty plugin skeleton is written to the file (if withfile is True)

(type=str)

Return Value

the path to the __init__.py of the new created plugin

(type=str)

removePlugin(*name*)

Uninstall a plugin (delete all data in scripts directory).

Parameters

`name:` the name of the plugin

(type=str)

installPlugin(*self*, *addon*, *data*)

Installs a new plugin into the scripts directory.

Parameters

`addon:` json dict containing the plugin information

(type=dict)

`data:` either the content of a single python file as string or a file-like-object to a zipfile which will be extracted

(type=str or file-like)

installPackages(*self*, *deps*)

Installs packages from pypi.python.org into the include directory.

Parameters

`deps:` A list of package names

(type=list[str])

Return Value

True on success, False otherwise

(type=bool)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(),

__repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

1.2.2 Properties

Name	Description
Inherited from object	
__class__	

1.2.3 Class Variables

Name	Description
PLUGIN_SKELETON	**Value:** ...

2 Module pluginhost

2.1 Functions

logprint(*msg*, *loglevel*, *channel*)

2.2 Variables

Name	Description
REL_URL	**Value:** `QUrl("https://api.github.com/repos/pathmann/pyTSon/releas.`

2.3 Class PluginHost

object
└
pytson.Translatable
└
pluginhost.PluginHost

2.3.1 Methods

setupConfig(*cls*)

verboseLog(*cls*, *text*, *channel*)

init(*cls*)

setupTranslator(*cls*)

startPlugin(*cls*, *key*)

start(*cls*)

shutdown(*cls*)

activate(*cls*, *pname*)

deactivate(*cls*, *pname*)

reload(*cls*)

showScriptingConsole(*cls*)

scriptingConsoleDestroyed(*cls*)

configure(*cls*, *mainwindow*=`None`)

callMethod(*cls*, *name*, **args*)

registerCallbackProxy(*cls*, *obj*)

unregisterCallbackProxy(*cls*, *obj*)

processCommand(*cls*, *schid*, *command*)

infoData(*cls*, *schid*, *aid*, *atype*)

parseUpdateReply(*cls*, *repstr*)

updateCheckFinished(*cls*, *reply*)

updateCheck(*cls*)

initMenus(*cls*)

globalMenuID(*cls*, *plugin*, *localid*)

initHotkeys(*cls*)

onMenuItemEvent(*cls*, *schid*, *atype*, *menuItemID*, *selectedItemID*)

globalHotkeyKeyword(*cls*, *plugin*, *localkeyword*)

onHotkeyEvent(*cls*, *keyword*)

showChangelog(*cls*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __init__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

2.3.2 Properties

Name	Description
Inherited from object	
__class__	

2.3.3 Class Variables

Name	Description
defaultConfig	**Value:** `[("general", [("differentApi", "False"), ("uninstallQuest...`

3 Module pylupdate

```
pylupdate parses python files to search for calls to pytson.tr resp
pytson.Translatable._tr to generate a Qt linguist translation file

These are the two valid usecases from module pytson:

class Translatable(object):
    @classmethod
    def _tr(cls, sourcetext, *, disambiguation="", n=-1, context=None):
        pass

def tr(context, sourcetext, *, disambiguation="", n=-1):
        pass
```

3.1 Functions

getSourceTexts(*inputfile*)

Extracts all translate function info from a python source file or Qt ui file

Parameters

`inputfile`: path to the python source file/ui file

(type=str)

Return Value

a list of tuples, containing the contextname and the message

(type=list[tuple(str, Message)])

main(*argv*)

Main function of pylupdate

Parameters

`argv`: the arguments passed to the scripts

(type=list[str])

3.2 Variables

Name	Description
DTD	**Value:** `r"""<...`

3.3 Class Message

object
└
pylupdate.Message

Wrapper for a translated message

3.3.1 Methods

__init__(*self*, *sourcetext*, *disambiguation*, *nused*, *finished*=`False`)

Instantiates a new Message object

Parameters

`sourcetext:`	sourcetext *(type=str)*
`disambiguation:`	string to distinguish between two equal sourcetexts *(type=str or None)*
`nused:`	if True, numerous translations will be used *(type=bool)*
`finished:`	defines, whether the translation is finished. defaults to False *(type=bool)*

Overrides: object.__init__

sourcetext(*self*, *val*)

disambiguation(*self*, *val*)

isNumerous(*self*, *val*)

isFinished(*self*, *val*)

__iter__(*self*)

Yields the translations of the Message

Return Value

the translated string(s)

(type=str)

__repr__(*self*)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

setTranslation(*self*, *trans*, *numeroustrans*=`None`)

Sets the translation of the Message

Parameters

`trans`: the translation

(type=str)

`numeroustrans`: if not None, the numerous translation

(type=str or None)

update(*self*, *msg*)

Updates the properties with the ones from another message

Parameters

`msg`: the other message

(type=Message)

fromXml(*elem*)

Parses the xml element message to a Message object

Parameters

`elem`: the xml element

(type=ElementTree.Element)

Return Value

a new created message object

(type=Message)

toXml(*self*)

Creates the xml elements of the message

Return Value

the xml element

(type=ElementTree.Element)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

3.3.2 Properties

Name	Description
Inherited from object	
__class__	

3.4 Class Context

object

pylupdate.Context

Wrapper for a translation context

3.4.1 Methods

__init__(*self*)

Instantiates a new Context object

Overrides: object.__init__

__iter__(*self*)

Yields each Message object in the context

Return Value

the message objects

(type=Message)

__repr__(*self*)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

addMessage(*self*, *msg*)

Adds a message to the context. If already exists, updates it.

Parameters

`msg`: the message

(type=Message)

update(*self*, *ctx*)

Updates all current messages with data from another context (if contained).

Parameters

`ctx`: the other context

(type=Context)

fromXml(*elem*)

Creates a context object from the xml element

Parameters

`elem`: the xml element

(type=ElementTree.Element)

Return Value

a tuple containing the name and the new context object

(type=tuple(str, Context))

toXml(*self*, *name*)

Creates the xml elements of the context

Parameters

`name`: the name of the context

(type=str)

Return Value

the xml element

(type=ElementTree.Element)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

3.4.2 Properties

Name	Description
Inherited from object	
__class__	

3.5 Class Translation

object
└
pylupdate.Translation

Wrapper for a Qt linguist translation file

3.5.1 Methods

__init__(*self*, *filename*, *language*)

Instantiates a new Translation object

Parameters

`filename`: path to read from resp. write to
(type=str)

`language`: the target language code
(type=str)

Overrides: object.__init__

read(*self*, *filename*=`None`)

Read the translation file given by the filename

Parameters

`filename`: if given, the path to the file to read. Defaults to None
(type=str)

write(*self*, *language*, *dtd*=`None`, *filename*=`None`)

Writes the data to a file.

Parameters

`language:` the target language code

(type=str)

`dtd:` if set, the xml is validated before written, defaults to None, throws Exception if validation failed

(type=str)

`filename:` if given, the path to write to, defaults to None

(type=str)

__**contains**__(*self*, *key*)

Checks, if a context is contained

Parameters

`key:` the context name

(type=str)

Return Value

returns True, if a context is contained by key's name

(type=bool)

__**getitem**__(*self*, *key*)

Returns the context object references by its name

Parameters

`key:` the name of the context

(type=str)

Return Value

the context

(type=Context)

__**iter**__(*self*)

Yields each context name contained in the translation

Return Value

the context names

(type=str)

addContext(*self*, *key*)

Creates a new context. A previous context with that name will be overwritten

Parameters

`key`: the name of the context

(type=str)

removeContext(*self*, *key*)

Remove a context.

Parameters

`key`: the name of the context

(type=str)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

3.5.2 Properties

Name	Description
Inherited from object	
__class__	

3.6 Class ParentVisitor

object

ast.NodeVisitor

pylupdate.ParentVisitor

ast Visitor which sets links to the parent node

3.6.1 Methods

generic_visit(*self*, *node*)

Called if no explicit visitor function exists for a node.

Overrides: ast.NodeVisitor.generic_visit extit(inherited documentation)

Inherited from ast.NodeVisitor

visit()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __init__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

3.6.2 Properties

Name	Description
Inherited from object	
__class__	

3.7 Class FunctionValidator

object

ast.NodeVisitor

pylupdate.ParentVisitor

pylupdate.FunctionValidator

ast Visitor to find calls to pyTSon's translation functions

3.7.1 Methods

__init__(*self*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

visit_Import(*self*, *node*)

Visits each import of the ast

Parameters

`node`: the import node

(type=ast.Import)

visit_ImportFrom(*self*, *node*)

Visits each import (from) of the ast

Parameters

`node`: the import node

(type=ast.ImportFrom)

visit_Call(*self*, *node*)

Visits each function call of the ast

Parameters

`node`: the call node

(type=ast.Call)

Inherited from pylupdate.ParentVisitor(Section 3.6)

generic_visit()

Inherited from ast.NodeVisitor

visit()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

3.7.2 Properties

Name	Description
Inherited from object	
__class__	

4 Module pythonqtpytson

Module PythonQt.pytson

4.1 Class EventFilterObject

PythonQt.QtCore.QObject
└ **pythonqtpytson.EventFilterObject**

Class to install as eventfilter on QObject-based objects. Connect to the signal eventFiltered(QObject*, QEvent*) to receive the filtered event per installEventFilter.

4.1.1 Methods

__init__(*self*, *typelist*=`list()`, *parent*=`None`)

Instantiates a new object.

Parameters

`typelist:` list of Eventtypes to filter (see QEvent::Type). Defaults to an empty list.

(type=list(int))

`parent:` QObject-parent

(type=QObject)

setFilterResult(*self*, *val*)

Sets the return value the object should return in the eventFilter-method. If not set, False will be returned

Parameters

`val:` the value

(type=bool)

types(*self*)

Returns the current list of eventtypes.

Return Value

the current eventtypes

(type=list(int))

addType(*self*, *eventtype*)

Adds an eventtype to the list.

Parameters

`eventtype:` *(type=an eventtype (see QEvent::Type))*

removeType(*self*, *eventtype*)

Removes an eventtype from the list.

Parameters

`eventtype:` an eventtype (see QEvent::Type)

(type=int)

5 Module pytson

5.1 Functions

tr(*context*, *sourcetext*, *disambiguation*="", *n*=-1)

Returns the current translation for a string. This function calls can be extracted by pyTSon's pylupdate.py.

Parameters

`context:` context of the string literal, must be a raw string, not the return value of another function, an attribute or such

(type=str)

`sourcetext:` translatable string, must be a raw string, not the return value of another function, an attribute or such

(type=str)

`disambiguation:` used to distinguish between two equal sourcetexts int the same context, or as comment, optional, defaults to an empty string, must be a raw string, not the return value of another function, an attribute or such

(type=str)

`n:` used for strings containing plurals, optional, defaults to -1

(type=int)

locales()

Generator function to return all locale codes available for translation in format language_country (see ISO 639 and ISO 3166)

Return Value

the language code

(type=str)

getConfigPath(**args*)

Returns pyTSon's configpath, that is, the subdirectory 'pyTSon' in the TeamSpeak 3 config directory.

Parameters

`args`: path fields joined to the result as list of strings

(type=list[str])

Return Value

The accumulated path

(type=str)

getPluginPath(**args*)

Returns pyTSon's pluginpath, that is, the subdirectory 'pyTSon' in the TeamSpeak 3 plugins directory.

Parameters

`args`: path fields joined to the result as list of strings

(type=list[str])

Return Value

The accumulated path

(type=str)

platformstr()

Returns the platform pyTSon is currently running on.

Return Value

the platform (and architecture) string

(type=str)

getVersion()

Returns the current version of pyTSon.

Return Value

the version as string

(type=str)

getCurrentApiVersion()

Returns the current apiversion of the ts3 plugin sdk. This should not be used in ts3plugin.apiVersion. Be fair and update your plugin manually!

Return Value

the apiVersion

(type=str)

5.2 Class Translatable

object
└ **pytson.Translatable**

Baseclass for a class using translatable string literals.

5.2.1 Methods

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __init__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

5.2.2 Properties

Name	Description
Inherited from object	
__class__	

6 Package pytsonui

6.1 Modules

- **config** *(Section 7, p. 28)*
- **console** *(Section 8, p. 31)*
- **dialogs** *(Section 9, p. 34)*
- **repository** *(Section 10, p. 35)*

6.2 Functions

ts3print(*msg*, *level*, *channel*, *aid*)

setIcon(*obj*, *iconpack*, *pluginicons*=`None`)

Sets the icon of a QWidget (if it has a property Icon) to an icon in the iconpack represented by a variable which is acquired by the property 'pytsonicon' of the object. If the property instead contains a string formated as "octicons:filename.png", the icon is set to filename.png of the octicons pack. If no such property is available, nothing is done.

Parameters

`obj:` the widget
(type=QWidget)

`iconpack:` the iconpack
(type=ts3client.IconPack)

`pluginicons:` callable which gets a string and either returns the path to the image file or returns a QPixmap to set the icon property to; defaults to None
(type=Callable(str) -> str or QIcon)

connectSignalSlotsByName(*sender*, *receiver*)

Connects pythonqt signals by name (receiver.on_<sender.objectname>_<signalname>)

Parameters

`sender:` the sender of signals
(type=QObject)

`receiver:` the receiver which has slots as callables defined
(type=object)

retrieveWidgets(*obj*, *parent*, *widgets*, *seticons*=`True`, *iconpack*=`None`, *pluginicons*=`None`)

Retrieves widgets from a list and adds them as attribute to another object. If defined, signals from widgets are connected by name to methods in obj.

Parameters

`obj`: the object which will get the attributes added

(type=object)

`parent`: the toplevel widget

(type=QWidget)

`widgets`: a recursive (parent-relation of widgets) list of tuples, defining which widgets should be added as attributes to obj. The elements must be children of parent. First element of tuple must held the widget's objectname. If second element is True, the widget will be added as property (by objectname) to obj. Third element of the tuple are the child widgets, which should be handled by setupui

(type=list[tuple(str, bool, list(...))])

`seticons`: if True, icons will be set according to the widgets 'pytsonicon' attribute

(type=bool)

`iconpack`: the iconpack

(type=ts3client.IconPack)

`pluginicons`: callable which gets a string and either returns the path to the image file or returns a QPixmap to set the icon property to; defaults to None

(type=Callable(str) -> str or QIcon)

retrieveAllWidgets(*obj*, *parent*, *seticons*=`True`, *iconpack*=`None`, *pluginicons*=`None`)

Retrieves all child widgets from a parent widget and adds them as attribute to another object. If defined, signals from widgets are connected by name to methods in obj.

Parameters

`obj:`	the object which will get the attributes added *(type=object)*
`parent:`	the toplevel widget *(type=QWidget)*
`seticons:`	if True, icons will be set according to the widgets 'pytsonicon' attribute *(type=bool)*
`iconpack:`	the iconpack *(type=ts3client.IconPack)*
`pluginicons:`	callable which gets a string and either returns the path to the image file or returns a QPixmap to set the icon property to; defaults to None *(type=Callable(str) -> str or QIcon)*

setupUi(*obj*, *uipath*, *widgets*=`None`, *seticons*=`True`, *iconpack*=`None`, *pluginicons*=`None`)

Loads a Qt designer file (.ui), creates the widgets defined in and adds them as property to a given object. This internally calls retrieveWidgets, so signals from widgets are connected by name to obj.

Parameters

`obj`: The object which will act as parent of the loaded ui (this object will receive a new layout)

(type=QWidget)

`uipath`: the path to the Qt designer file

(type=str)

`widgets`: optional argument; a recursive (parent-relation of widgets) list of tuples, defining which widgets should be added as attributes to obj. See retrieveWidgets for details. If you omit this or pass None, recursively all child widgets will be stored

(type=list[tuple(str, bool, list(...))] or None)

`seticons`: if True, widgets containing a string-property called 'pytsonicon' will get the icon of a soundpack (value of property = variable in soundpack)

(type=bool)

`iconpack`: if set, the iconpack will be used, if None, the current iconpack is used

(type=ts3client.IconPack)

`pluginicons`: callable which gets a string and either returns the path to the image file or returns a QPixmap to set the icon property to; defaults to None

(type=Callable(str) -> str or QIcon)

6.3 Class UiLoader

PythonQt.QtUiTools.QUiLoader
└ **pytsonui.UiLoader**

QUILoader subclass to omit the parent widget from being recreated.

6.3.1 Methods

__init__(*self*, *main*, *parent*=`None`)

Instantiate a new object

Parameters

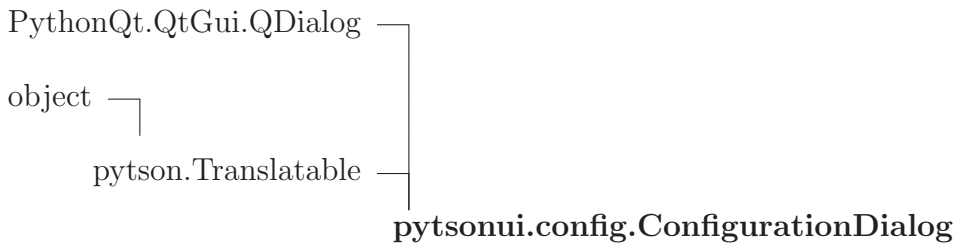
`main:` parent class which will be omitted
(type=QWidget)

`parent:` parent class; defaults to None
(type=QObject)

createWidget(*self*, *clsname*, *parent*=`None`, *name*=`''`)

7 Module pytsonui.config

7.1 Class ConfigurationDialog



7.1.1 Methods

__init__(*self*, *cfg*, *host*, *parent*=`None`)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

setupList(*self*)

setupValues(*self*)

setupSlots(*self*)

onLoadMenusButtonChanged(*self*, *state*)

onDifferentApiButtonChanged(*self*, *state*)

onPluginsTableCurrentItemChanged(*self*, *currow*, *curcol*, *prevrow*, *prevcol*)

onPluginsTableItemChanged(*self*, *item*)

onRemoveButtonClicked(*self*, *pluginname*)

onReloadButtonClicked(*self*)

onSettingsButtonClicked(*self*, *pluginname*)

onBackgroundColorButtonClicked(*self*)

onTextColorButtonClicked(*self*)

onFontFamilyComboChanged(*self*, *font*)

onFontSizeSpinChanged(*self*, *size*)

onTabcompleteButtonChanged(*self*, *state*)

onSpacesButtonChanged(*self*, *state*)

onTabwidthSpinChanged(*self*, *width*)

on_scriptButton_clicked(*self*)

on_scriptEdit_textEdited(*self*, *text*)

on_silentButton_toggled(*self*, *act*)

on_repositoryButton_clicked(*self*)

onLanguageComboCurrentIndexChanged(*self*, *idx*)

onLanguageButtonStateChanged(*self*, *state*)

onVerboseButtonStateChanged(*self*, *state*)

reloadSite(*self*)

on_siteTable_itemSelectionChanged(*self*)

on_siteaddButton_clicked(*self*)

on_siteremoveButton_clicked(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

7.1.2 Properties

Name	Description
Inherited from object __class__	

7.1.3 Class Variables

Name	Description
CONF_WIDGETS	**Value:** `[("tabWidget", False, [("pluginsTab", False, [("different...`

8 Module pytsonui.console

8.1 Functions

defaultFont()

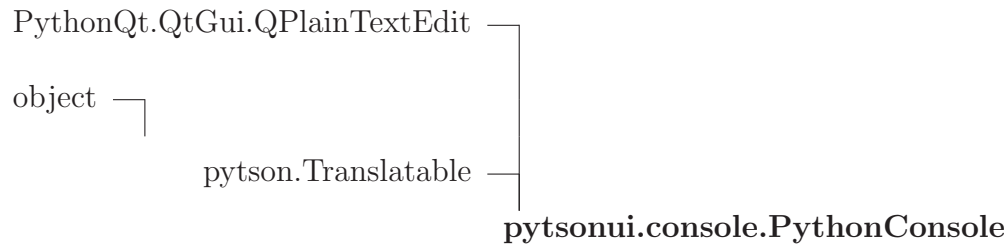
8.2 Class StdRedirector

8.2.1 Methods

__**init**__(*self*, *callback*)

write(*self*, *text*)

8.3 Class PythonConsole



8.3.1 Methods

__**init**__(*self*, *tabcomplete*=`True`, *spaces*=`True`, *tabwidth*=`2`, *font*=`defaultFont()`, *bgcolor*=`Qt.black`, *textcolor*=`Qt.white`, *width*=`800`, *height*=`600`, *startup*=`""`, *silentStartup*=`False`, *parent*=`None`, *catchstd*=`False`)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

setFont(*self*, *f*)

prompt(*self*)

promptLength(*self*)

writePrompt(*self*, *newline*)

promptCursor(*self*)

keyPressEvent(*self*, *e*)

mousePressEvent(*self*, *e*)

mouseReleaseEvent(*self*, *e*)

doKeyboardInterrupt(*self*)

doEndFile(*self*)

currentLine(*self*)

removeCurrentLine(*self*)

addHistory(*self*, *cmd*)

doHistoryUp(*self*)

doHistoryDown(*self*)

doTab(*self*)

doUntab(*self*)

appendLine(*self*, *text*)

runCommand(*self*, *cmd*, *silent*)

doExecuteCommand(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

8.3.2 Properties

Name	Description
Inherited from object	
__class__	

9 Module pytsonui.dialogs

9.1 Class MultiInputDialog

PythonQt.QtGui.QDialog
└ **pytsonui.dialogs.MultiInputDialog**

9.1.1 Methods

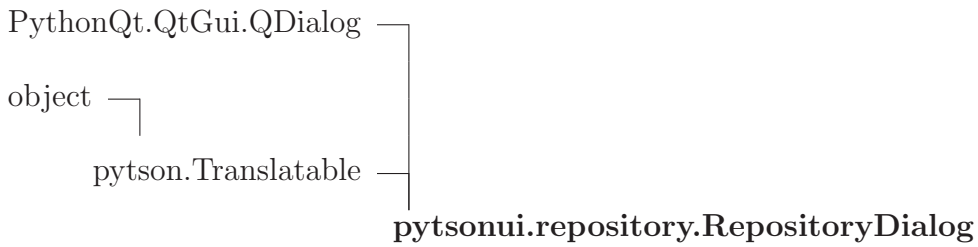
__init__(*self*, *title*, *label1*, *label2*, *parent*=`None`)

cleanup(*self*)

getTexts(*title*, *label1*, *label2*, *text1*=`""`, *text2*=`""`, *parent*=`None`)

10 Module pytsonui.repository

10.1 Class RepositoryDialog



10.1.1 Methods

__init__(*self*, *host*, *parent*=`None`)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

onClosed(*self*)

updatePendingButtons(*self*)

updateRepositories(*self*)

updateMaster(*self*)

handleMasterReply(*self*, *reply*)

updateAddons(*self*, *repo*, *addons*)

handleRepositoryReply(*self*, *reply*)

onNetworkReply(*self*, *reply*)

addRepository(*self*, *r*)

updateMasterlist(*self*)

updateAddonlist(*self*)

on_updateButton_clicked(*self*)

on_addButton_clicked(*self*)

on_deleteButton_clicked(*self*)

on_repositoryList_doubleClicked(*self*, *item*)

on_repositoryList_currentItemChanged(*self*, *cur*, *prev*)

on_repositoryList_itemChanged(*self*, *item*)

on_pluginsList_currentItemChanged(*self*, *cur*, *prev*)

on_reloadButton_clicked(*self*)

on_installButton_clicked(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

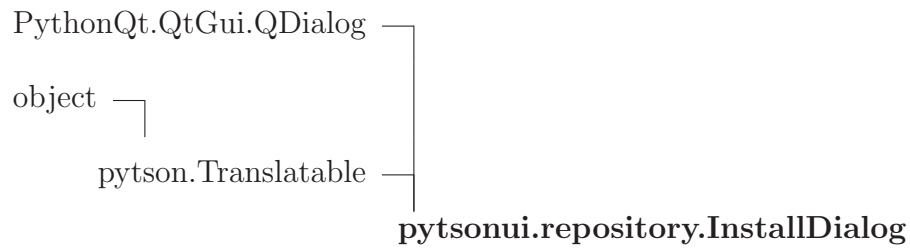
10.1.2 Properties

Name	Description
Inherited from object	
__class__	

10.1.3 Class Variables

Name	Description
master_url	**Value:** `QUrl("https://raw.githubusercontent.com/pathmann/pyTSon_r..`

10.2 Class InstallDialog



10.2.1 Methods

__init__(*self*, *host*, *parent*=`None`)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

install(*self*, *addon*)

installPackage(*self*, *pkgstr*)

onNetworkReply(*self*, *reply*)

on_closeButton_clicked(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

10.2.2 Properties

Name	Description
Inherited from object	
__class__	

11 Module signalslot

11.1 Class Signal

object
└ **signalslot.Signal**

Simple class to emit signals to connected callable receivers.

11.1.1 Methods

__init__(*self*)

Instantiate a new object

Overrides: object.__init__

connect(*self*, *c*)

Connect a callable as receiver for the signal

Parameters

`c`: signal receiver

(type=Callable)

disconnect(*self*, *c*)

Disconnect the callable from receiving the signal

Parameters

`c`: signal receiver

(type=Callable)

disconnectAll(*self*)

Disconnects all signal receivers

emit(*self*, **args*, ***kwargs*)

Fires the signal to all connected receivers

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

11.1.2 Properties

Name	Description
Inherited from object __class__	

12 Module ts3client

12.1 Class Config

object
└ **ts3client.Config**

Offers an interface to query the TeamSpeak 3 client's config database (settings.db). You should always del a reference to this object if not needed anymore to assure the database connection is closed.

12.1.1 Methods

__init__(*self*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__del__(*self*)

__getattr__(*self*, *name*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

12.1.2 Properties

Name	Description
Inherited from object	
__class__	

12.1.3 Class Variables

Name	Description
objcount	**Value:** `0`
instance	**Value:** `None`

12.2 Class IconPack

object
└ **ts3client.IconPack**

Offers an interface to the TeamSpeak 3 Client's iconpack. IconPack is also a context manager.

12.2.1 Methods

current()

Returns the current iconpack used (an Exception is raised if something failed).

Return Value

the iconpack

(type=IconPack)

__**init**__(*self*, *info*=`None`, *name*=`None`)

Instantiates a new IconPack object referenced by its name or the internal info string (an Exception is raised if the iconpack could not be located).

Parameters

`info`: the info string used in the settings.db

(type=str)

`name`: the name of the iconpack

(type=str)

Overrides: object.__init__

open(*self*)

Reads the settings for the iconpack and if it's zip-based, opens the file for reading. Must be called once before any icon can be accessed.

close(*self*)

If the iconpack is zip-based, the file is closed. After this is called, no icons can be accessed (till open is called again).

__**enter**__(*self*)

__**exit**__(*self*, *type*, *value*, *traceback*)

defaultName(*var*)

Returns the variable name used in the default iconpack.

Parameters

`var`: the variable used in an iconpack

(type=str)

Return Value

the variable name

(type=str)

fallback(*self*, *var*)

Returns the fallback icon for a variable according to the iconpack's settings.

Parameters

`var`: the variable name

(type=str)

Return Value

the resulting pixmap

(type=QPixmap)

icons(*self*)

Returns the list of variables used in the iconpack (excluding fallback mechanisms).

Return Value

a list of variable names

(type=list[str])

icon(*self*, *var*)

Returns the icon representing a variable used in the iconpack. If the icon cannot be found, the iconpack's fallback mechanisms are used. If everything fails, an empty pixmap is returned.

Parameters

`var`: the variable name

(type=str)

Return Value

the resulting pixmap

(type=QPixmap)

emoticons(*self*)

Returns the list of emoticon replacements used in the iconpack.

Return Value

a list of emoticon strings

(type=list[str])

emoticon(*self*, *text*)

Returns the icon replacing the emoticon string.

Parameters

`text`: the emoticon as string

(type=str)

Return Value

the resulting pixmap

(type=QPixmap)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

12.2.2 Properties

Name	Description
Inherited from object	
__class__	

12.3 Class ServerCache

Offers an interface to the cached data of a TeamSpeak 3 server.

12.3.1 Methods

__init__(*self*, *schid*)

Instantiates a new ServerCache object referenced by the server connection handler id (an Exception is raised if the path in the filesystem could not be located).

Parameters

`schid`: the ID of the serverconnection

(type=int)

icon(*self*, *iconid*)

Returns an icon cached on disk.

Parameters

`iconid`: ID of the icon

(type=int)

Return Value

the icon

(type=QPixmap)

12.4 Class CountryFlags

Offers an interface to get the client's country flags. CountryFlags is also a context manager.

12.4.1 Methods

__init__(*self*)

Instantiates a new object. This will raise an exception, if the Zipfile could not be located.

open(*self*)

Opens the Zipfile for reading. This must be called before any flag is requested with flag.

close(*self*)

Closes the Zipfile.

flag(*self*, *code*)

Returns a QPixmap containing the flag of the given country code if exist.

Parameters

`code:` the country code

Return Value

the flag

(type=QPixmap)

__**enter**__(*self*)

__**exit**__(*self*, *type*, *value*, *traceback*)

13 Module ts3lib

13.1 Functions

getPluginID()

Returns pyTSon's plugin id

Return Value

the plugin id

(type=string)

acquireCustomPlaybackData(*deviceName*, *samples*)

Retrieves playback data from the clientlib

Parameters

`deviceName:` the name of the playback device previously registered with registerCustomDevice

(type=string)

`samples:` specifies how long the resultbuffer should be, which is passed to the clientlib

(type=int)

Return Value

a tuple containing the errorcode and the buffer as list of ints

(type=tuple(int, list[int]))

activateCaptureDevice(*serverConnectionHandlerID*)

Activates the capture device on a serverconnection.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection

(type=int)

Return Value

the errorcode

(type=int)

banadd(*serverConnectionHandlerID*, *ipRegExp*, *nameRegexp*, *uniqueIdentity*, *timeInSeconds*, *banReason*, *returnCode*)

Adds a new ban.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`ipRegExp:`	regular expression to match IPs, pass an empty string to ignore IPs *(type=string)*
`nameRegexp:`	regular expression to match client nicknames, pass an empty string to ignore nicknames *(type=string)*
`uniqueIdentity:`	client UID to ban, pass an empty string to ignore UIDs *(type=string)*
`timeInSeconds:`	the time, the client should be banned for, pass 0 to add a permanent ban *(type=int)*
`banReason:`	the reason for the ban *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

banclient(*serverConnectionHandlerID*, *clientID*, *timeInSeconds*, *banReason*, *returnCode*)

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the ID of the client *(type=int)*
`timeInSeconds:`	the time, the client should be banned for, pass 0 to add a permanent ban *(type=int)*
`banReason:`	the reason for the ban *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

banclientdbid(*serverConnectionHandlerID*, *clientDBID*, *timeInSeconds*, *banReason*, *returnCode*)

Bans a user defined by his database ID.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientDBID:`	the database ID of the user *(type=int)*
`timeInSeconds:`	the time, the client should be banned for, pass 0 to add a permanent ban *(type=int)*
`banReason:`	the reason for the ban *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

bandel(*serverConnectionHandlerID*, *banID*, *returnCode*)

Deletes a ban.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`banID:`	the ID of the ban *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

bandelall(*serverConnectionHandlerID*, *returnCode*)

Deletes all bans on a server.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

channelPropertyStringToFlag(*channelPropertyString*)

Converts a channel property name used in strings (eg the serverquery) to the corresponding flag.

Parameters

`channelPropertyString:` *(type=string)*

Return Value

a tuple, containing the errorcode and the flag (see ts3defines.ChannelProperties and ts3defines.ChannelPropertiesRare)

(type=tuple (int, int))

channelset3DAttributes(*serverConnectionHandlerID*, *clientID*, *position*)

Adjusts a clients position and velocity in 3D space.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the ID of the client to adjust
(type=int)

`position:` a tuple defining the position of the clientID
(type=tuple (float, float, float))

Return Value

the errorcode

(type=int)

cleanUpConnectionInfo(*serverConnectionHandlerID*, *clientID*)

//FIXME:

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the ID of the client
(type=int)

Return Value

the errorcode

(type=int)

clientChatClosed(*serverConnectionHandlerID*, *clientUniqueIdentifier*, *clientID*, *returnCode*)

Sends the client chat closed command to a client the own client is currently chatting with.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientUniqueIdentifier:`	the uid of the own chatting client *(type=string)*
`clientID:`	the ID of the client, the own client is chatting with *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

clientChatComposing(*serverConnectionHandlerID*, *clientID*, *returnCode*)

Sends the client chat composing command to a client the own client is currently chatting with.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the ID of the client, the own client is chatting with
(type=int)

`returnCode:` returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional.
(type=string)

Return Value

the errorcode
(type=int)

clientPropertyStringToFlag(*clientPropertyString*)

Converts a client property name used in strings (eg the serverquery) to the corresponding flag.

Parameters

`clientPropertyString:` *(type=string)*

Return Value

a tuple, containing the errorcode and the flag (see ts3defines.ClientProperties and ts3defines.ClientPropertiesRare)
(type=tuple (int, int))

closeCaptureDevice(*serverConnectionHandlerID*)

Closes a capture device on a serverconnection.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

Return Value

the errorcode
(type=int)

closePlaybackDevice(*serverConnectionHandlerID*)

Closes a playback device on a serverconnection.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection

(type=int)

Return Value

the errorcode

(type=int)

closeWaveFileHandle(*serverConnectionHandlerID*, *waveHandle*)

Closes a wavefile sound handle previously returned by playWaveFileHandle.

Parameters

`serverConnectionHandlerID:` the ID of the serverConnection the sound was played on

(type=int)

`waveHandle:` the handle returned by playWaveFileHandle

(type=int)

Return Value

the errorcode

(type=int)

createBookmark(*bookmarkuuid*, *serverLabel*, *serverAddress*, *serverPassword*, *nickname*, *channel*, *channelPassword*, *captureProfile*, *playbackProfile*, *hotkeyProfile*, *soundProfile*, *uniqueUserId*, *oneTimeKey*, *phoneticName*)

Creates a new bookmark.

Parameters

`serverLabel:` the label of the connection
(type=string)

`serverAddress:` host or ip address
(type=string)

`serverPassword:` password to the server, pass an empty string if the server is not password protected
(type=string)

`nickname:` the user's nickname
(type=string)

`channel:` complete path to the channel to connect to
(type=string)

`channelPassword:` password to the channel, pass an empty string if the channel is not password protected
(type=string)

`captureProfile:` the name of the capture profile to use; pass an empty string to always use the default one
(type=string)

`playbackProfile:` the name of the playback profile to use; pass an empty string to always use the default one
(type=string)

`hotkeyProfile:` the name of the hotkey profile to use; pass an empty string to always use the default one
(type=string)

`soundProfile:` the name of the sound profile to use; pass an empty string to always use the default one
(type=string)

`uniqueUserId:` identity (name) to use; pass an empty string to always use the default one
(type=string)

`oneTimeKey:` privilege key to use on connect
(type=string)

`phoneticName:` phonetic nickname
(type=string)

Return Value

the errorcode

createReturnCode(*maxLen*=128)

Creates a returnCode which can be passed to the other functions and will be passed to the event onServerErrorEvent.

Parameters

`maxLen`: length of the buffer, passed to the clientlib to store the path to, default value is 256

(type=int)

Return Value

the created returnCode

(type=string)

destroyServerConnectionHandler(*serverConnectionHandlerID*)

Destroys a server connection handler.

Parameters

`serverConnectionHandlerID`: the ID of the serverconnection

(type=int)

Return Value

the errorcode

(type=int)

flushChannelCreation(*serverConnectionHandlerID*, *channelParentID*, *returnCode*)

Flushes the channel creation made by the setChannelVariable-functions to the server.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelParentID:`	the ID of the parent channel of the new channel *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

flushChannelUpdates(*serverConnectionHandlerID*, *channelID*, *returnCode*)

Flushes the changes made by the setChannelVariable-functions to the server.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channelID *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

flushClientSelfUpdates(*serverConnectionHandlerID*, *returnCode*)

Flushes the changes made by the setClientSelfVariable-functions to the server.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`returnCode:` returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional.
(type=string)

Return Value

the errorcode
(type=int)

getAppPath(*maxLen*=`256`)

Returns the ts3 application path.

Parameters

`maxLen:` length of the buffer, passed to the clientlib to store the path to, default value is 256
(type=int)

Return Value

the application path
(type=string)

getAvatar(*serverConnectionHandlerID*, *clientID*, *maxLen*=256)

Returns the path on the system to the avatar image file of a client.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the ID of the client
(type=int)

`maxLen:` length of the buffer, passed to the clientlib to store the path to, default value is 256
(type=int)

Return Value

a tuple, containing the errorcode and the path to the avatar
(type=tuple (int, string))

getAverageTransferSpeed(*transferID*)

Returns the average transfer speed of a filetransfer.

Parameters

`transferID:` the ID of the filetransfer
(type=int)

Return Value

a tuple, containing the errorcode and the speed
(type=tuple (int, float))

getBookmarkList()

Returns the list of bookmarks.

Return Value

a tuple, containing the errorcode and a list of tuples (name, isFolder, uid, childs)
(type=tuple (int, [tuple (string, int or bool, string or None, [childs])]))

getCaptureDeviceList(*modeID*)

Queries all available capture devices.

Parameters

`modeID:` Defines the capture mode to use.

(type=string)

Return Value

A tuple, containing the errorcode and the list of capture devices as tuple (devicename, deviceid)

(type=tuple (int, [(string, string)]))

getCaptureModeList()

Queries all available capture modes.

Return Value

A tuple, containing the errorcode and the list of capture modes

(type=tuple (int, [string]))

getChannelClientList(*serverConnectionHandlerID*, *channelID*)

Returns all clients in a specified channel.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection

(type=int)

`channelID:` the ID of the channel

(type=int)

Return Value

a tuple, containing the errorcode and a list of client IDs or None if the call failed

(type=tuple (int, [int]) or tuple(int, None))

getChannelConnectInfo(*serverConnectionHandlerID*, *channelID*, *maxLen*)

Returns the channel connect info (path and password) of a channel.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelID:` the ID of the channel
(type=int)

`maxLen:` length of the buffer, passed to the clientlib to store the path to, default value is 256
(type=int)

Return Value

a tuple, containing the errorcode, the path and the password of a channel
(type=tuple (int, string, string))

getChannelIDFromChannelNames(*serverConnectionHandlerID*, *channelNameArray*)

Returns the ID of a channel defined by its name.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelNameArray:` list of strings, defining the position of the channel (['grandparent', 'parent', 'channel'])
(type=list [string])

Return Value

a tuple, containing the errorcode and the ID of the channel
(type=tuple (int, int))

getChannelList(*serverConnectionHandlerID*)

Returns all channels on the server.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

Return Value

a tuple, containing the errorcode and a list of channel IDs
(type=tuple (int, [int]))

getChannelOfClient(*serverConnectionHandlerID*, *clientID*)

Returns the channel of a client.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the ID of the client
(type=int)

Return Value

a tuple, containing the errorcode and the channel
(type=tuple (int, int))

getChannelVariableAsInt(*serverConnectionHandlerID*, *channelID*, *flag*)

Returns a channel variable as int value.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelID:` the ID of the channel
(type=int)

`flag:` the flag to return
(type=int)

Return Value

a tuple, containing the errorcode and the value of the flag
(type=tuple (int, int))

getChannelVariableAsString(*serverConnectionHandlerID*, *channelID*, *flag*)

Returns a channel variable as string value.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelID:` the ID of the channel
(type=int)

`flag:` the flag to return
(type=int)

Return Value

a tuple, containing the errorcode and the value of the flag
(type=tuple (int, int))

getChannelVariableAsUInt64(*serverConnectionHandlerID*, *channelID*, *flag*)

Returns a channel variable as unsigned long long int value.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelID:` the ID of the channel
(type=int)

`flag:` the flag to return
(type=int)

Return Value

a tuple, containing the errorcode and the value of the flag
(type=tuple (int, int))

getClientDisplayName(*serverConnectionHandlerID*, *clientID*, *maxLen*=128)

Returns the client display name receiving from the client's contacts settings.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the ID of the client
(type=int)

`maxLen:` length of the buffer, passed to the clientlib to store the path to, default value is 128
(type=int)

Return Value

a tuple, containing the errorcode and the display name
(type=tuple (int, string))

getClientID(*serverConnectionHandlerID*)

Returns the own client ID on a given serverconnection.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

Return Value

a tuple, containing the errorcode and the client ID
(type=tuple (int, int))

getClientLibVersion()

Returns the clientlib's version as string.

Return Value

A tuple, containing the errorcode and the result
(type=tuple (int, string))

getClientLibVersionNumber()

Returns the clientlib's version number

Return Value

A tuple, containing the errorcode and the result

(type=tuple (int, int))

getClientList(*serverConnectionHandlerID*)

Returns all clients in view on the server.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

Return Value

a tuple, containing the errorcode and the list of client IDs

(type=tuple (int, [int]))

getClientNeededPermission(*serverConnectionHandlerID*, *permissionName*)

Returns the value of the client's needed permission.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`permissionName:` name of the permission
(type=string)

Return Value

a tuple, containing the errorcode and the value of the permission

(type=tuple (int, int))

getClientSelfVariableAsInt(*serverConnectionHandlerID*, *flag*)

Returns the value of a given flag of the own client as int value.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`flag:` the flag to return
(type=int)

Return Value

a tuple, containing the errorcode and the value of the queried flag
(type=tuple (int, int))

getClientSelfVariableAsString(*serverConnectionHandlerID*, *flag*)

Returns the value of a given flag of the own client as string.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`flag:` the flag to return
(type=int)

Return Value

a tuple, containing the errorcode and the value of the queried flag
(type=tuple (int, string))

getClientVariableAsInt(*serverConnectionHandlerID*, *clientID*, *flag*)

Returns the value of a given flag of a client as int value.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the ID of the client
(type=int)

`flag:` the flag to return
(type=int)

Return Value

a tuple, containing the errorcode and the value of the flag
(type=tuple (int, int))

getClientVariableAsString(*serverConnectionHandlerID*, *clientID*, *flag*)

Returns the value of a given flag of a client as string value.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the ID of the client
(type=int)

`flag:` the flag to return
(type=int)

Return Value

a tuple, containing the errorcode and the value of the flag
(type=tuple (int, string))

getClientVariableAsUInt64(*serverConnectionHandlerID*, *clientID*, *flag*)

Returns the value of a given flag of a client as unsigned long long int value.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the ID of the client
(type=int)

`flag:` the flag to return
(type=int)

Return Value

a tuple, containing the errorcode and the value of the flag
(type=tuple (int, int))

getConfigPath(*maxLen*=256)

Returns the ts3 config path.

Parameters

`maxLen:` length of the buffer, passed to the clientlib to store the path to, default value is 256
(type=int)

Return Value

the config path
(type=string)

getConnectionStatus(*serverConnectionHandlerID*)

Returns the current connection status of a serverconnection.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

Return Value

a tuple, containing the errorcode and the connection status
(type=tuple (int, int))

getConnectionVariableAsDouble(*serverConnectionHandlerID*, *clientID*, *flag*)

Returns a client's connection variable as python floating point variable.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the ID of the client
(type=int)

Return Value

a tuple, containing the errorcode and the value of the flag
(type=tuple (int, double))

getConnectionVariableAsString(*serverConnectionHandlerID*, *clientID*, *flag*)

Returns a client's connection variable as string variable.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the ID of the client
(type=int)

Return Value

a tuple, containing the errorcode and the value of the flag
(type=tuple (int, string))

getConnectionVariableAsUInt64(*serverConnectionHandlerID*, *clientID*, *flag*)

Returns a client's connection variable as unsigned long long int variable.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the ID of the client
(type=int)

Return Value

a tuple, containing the errorcode and the value of the flag
(type=tuple (int, int))

getCurrentCaptureDeviceName(*serverConnectionHandlerID*)

Queries the current playback device's name on a serverconnection.

Parameters

`serverConnectionHandlerID:` ID of the serverconnection
(type=int)

Return Value

A tuple, containing the errorcode, the capture device's name and the status, if it's default
(type=tuple (int, string, int))

getCurrentCaptureMode(*serverConnectionHandlerID*)

Queries the current capture mode on a serverconnection.

Parameters

`serverConnectionHandlerID:` ID of the serverconnection
(type=int)

Return Value

A tuple, containing the errorcode and the current capture mode
(type=tuple (int, string))

getCurrentPlaybackDeviceName(*serverConnectionHandlerID*)

Queries the current playback device's name on a serverconnection.

Parameters

`serverConnectionHandlerID:` ID of the serverconnection

(type=int)

Return Value

A tuple, containing the errorcode, the playback device's name and the status, if it's default

(type=tuple (int, string, int))

getCurrentPlayBackMode(*serverConnectionHandlerID*)

Queries the current playback mode on a serverconnection.

Parameters

`serverConnectionHandlerID:` ID of the serverconnection

(type=int)

Return Value

A tuple, containing the errorcode and the current playback mode

(type=tuple (int, string))

getCurrentServerConnectionHandlerID()

Returns the current serverconnection handler.

Return Value

the ID of the current serverconnection handler

(type=int)

getCurrentTransferSpeed(*transferID*)

Returns the current transfer speed of a filetransfer.

Parameters

`transferID:` the ID of the filetransfer

(type=int)

Return Value

a tuple, containing the errorcode and the speed

(type=tuple (int, float))

getDefaultCaptureDevice(*modeID*)

Queries the default capture device.

Parameters

`modeID`: Defines the capture mode to use

(type=string)

Return Value

A tuple, containing the errorcode and the default capture device as tuple (devicename, deviceid)

(type=tuple (int, (string, string)))

getDefaultCaptureMode()

Queries the default capture mode.

Return Value

A tuple, containing the errorcode and the default capture mode

(type=tuple (int, string))

getDefaultPlaybackDevice(*modeID*)

Queries the default playback device.

Parameters

`modeID`: Defines the playback mode to use

(type=string)

Return Value

A tuple, containing the errorcode and the default playback device as tuple (devicename, deviceid)

(type=tuple (int, (string, string)))

getDefaultPlayBackMode()

Queries the default playback mode.

Return Value

A tuple, containing the errorcode and the default playback mode

(type=tuple (int, string))

getDirectories(*path*, *maxLen*=256)

Returns a list of subdirectories of a path as space-separated string.

Parameters

`path:` the parent path

(type=str)

`maxLen:` length of the buffer, passed to the clientlib to store the path to, default value is 256

(type=int)

Return Value

the resulting path

(type=string)

getEncodeConfigValue(*serverConnectionHandlerID*, *ident*)

Queries a speex encoder option.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection

(type=int)

`ident:` the encoder option to be queried

(type=string)

Return Value

A tuple, containing the errorcode and the flag's value

(type=tuple (int, string))

getErrorMessage(*errorCode*)

Queries a printable error string for a specific error code.

Parameters

`errorCode:` The error code returned from all Client Lib functions

(type=int)

Return Value

A tuple, containing the errorcode and the resulting string

(type=tuple (int, string))

getHotkeyFromKeyword(*keywords*)

Returns a list of hotkeys by its keywords.

Parameters

`keywords:` a list of keywords
(type=list[str])

Return Value

a tuple containing the errorcode and the list of hotkeys
(type=tuple(int, list[str]))

getParentChannelOfChannel(*serverConnectionHandlerID*, *channelID*)

Returns the parent channel of another channel.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelID:` the ID of the channel
(type=int)

Return Value

a tuple, containing the errorcode and the ID of the parent channel
(type=tuple (int, int))

getPermissionIDByName(*serverConnectionHandlerID*, *permissionName*)

Returns the ID of a permission defined by its name.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`permissionName:` name of the permission
(type=string)

Return Value

a tuple, containing the errorcode and the ID of the permission
(type=tuple (int, int))

getPlaybackConfigValueAsFloat(*serverConnectionHandlerID*, *ident*)

Queries a playback option.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`ident:` the playback option to be queried
(type=string)

Return Value

A tuple, containing the errorcode and the flag's value
(type=tuple (int, float))

getPlaybackDeviceList(*modeID*)

Queries all available playback devices.

Parameters

`modeID:` Defines the playback mode to use.
(type=string)

Return Value

A tuple, containing the errorcode and the list of playback devices as tuple (devicename, deviceid)
(type=tuple (int, [(string, string)]))

getPlaybackModeList()

Queries all available playback modes.

Return Value

A tuple, containing the errorcode and the list of modes
(type=tuple (int, [string]))

getPluginPath(*path*, *maxLen*, *pluginID*)

Returns the TeamSpeak 3 client's pluginpath.

Parameters

`maxLen:` the size of the buffer passed to the clientlib. Optional, defaults to 256

(type=int)

Return Value

the pluginpath

(type=str)

getPreProcessorInfoValue(*serverConnectionHandlerID*, *ident*)

Querie a sound preprocessor flag and returns it as string.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection

(type=int)

`ident:` the flag to be queried

(type=string)

Return Value

A tuple, containing the errorcode and the value of the queried flag

(type=tuple (int, string))

getPreProcessorInfoValueFloat(*serverConnectionHandlerID*, *ident*)

Queries a sound preprocessor flag and returns it as float.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection

(type=int)

`ident:` the flag to be queried

(type=string)

Return Value

A tuple, containing the errorcode and the value of the queried flag

(type=tuple (int, float))

getProfileList(*profile*)

Returns a list of existing profiles and the default profile's index in list.

Parameters

`profile:` the profile type, see ts3defines.PluginGuiProfile

(type=int)

Return Value

a tuple, containing the errorcode, the default profile's index and the profile list

(type=tuple (int, int, [string]))

getResourcesPath(*maxLen*=`256`)

Returns the ts3 resources path.

Parameters

`maxLen:` length of the buffer, passed to the clientlib to store the path to. Optional, defaults to 256

(type=int)

Return Value

the resources path

(type=string)

getServerConnectInfo(*serverConnectionHandlerID*, *maxLen*=`256`)

Returns the connect info (host, port and password) of a serverconnection.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection

(type=int)

`maxLen:` length of the buffer, passed to the clientlib to store the path to, default value is 256

(type=int)

Return Value

a tuple, containing the errorcode, the host, the port and the password of the serverconnection

(type=tuple (int, string, int, string))

getServerConnectionHandlerList()

Returns a list of serverconnection handlers.

Return Value

a tuple, containing the errorcode and the list of serverconnection handler IDs

(type=tuple (int, [int]))

getServerVariableAsInt(*serverConnectionHandlerID*, *flag*)

Returns a server variable as int value.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`flag:` the flag to return
(type=int)

Return Value

a tuple, containing the errorcode and the value of the flag

(type=tuple (int, int))

getServerVariableAsString(*serverConnectionHandlerID*, *flag*)

Returns a server variable as string value.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`flag:` the flag to return
(type=int)

Return Value

a tuple, containing the errorcode and the value of the flag

(type=tuple (int, string))

getServerVariableAsUInt64(*serverConnectionHandlerID*, *flag*)

Returns a server variable as unsigned long long int value.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`flag:` the flag to return
(type=int)

Return Value

a tuple, containing the errorcode and the value of the flag
(type=tuple (int, int))

getServerVersion(*serverConnectionHandlerID*)

Returns the server version.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

Return Value

the server version
(type=int)

getTransferFileName(*transferID*)

Returns the filename of a filetransfer.

Parameters

`transferID:` the ID of the filetransfer
(type=int)

Return Value

a tuple, containing the errorcode and the filename
(type=tuple (int, string))

getTransferFilePath(*transferID*)

Returns the filepath of a filetransfer.

Parameters

`transferID`: the ID of the filetransfer

(type=int)

Return Value

a tuple, containing the errorcode and the filepath

(type=tuple (int, string))

getTransferFileSize(*transferID*)

Returns the total filesize (in Bytes) of a filetransfer.

Parameters

`transferID`: the ID of the filetransfer

(type=int)

Return Value

a tuple, containing the errorcode and the filesize

(type=tuple (int, int))

getTransferFileSizeDone(*transferID*)

Returns the already downloaded size (in Bytes) of a filetransfer.

Parameters

`transferID`: the ID of the filetransfer

(type=int)

Return Value

a tuple, containing the errorcode and the size

(type=tuple (int, int))

getTransferRunTime(*transferID*)

Returns the runtime of a filetransfer.

Parameters

`transferID`: the ID of the filetransfer

(type=int)

Return Value

a tuple, containing the errorcode and the runtime in seconds

(type=tuple (int, int))

getTransferStatus(*transferID*)

Returns the status of a filetransfer, whether if it is initialising, active or finished see ts3defines.FileTransferState

Parameters

`transferID:` the ID of the filetransfer

(type=int)

Return Value

a tuple, containing the errorcode and the status

(type=tuple (int, int))

guiConnect(*connectTab*, *serverLabel*, *serverAddress*, *serverPassword*, *nickname*, *channel*, *channelPassword*, *captureProfile*, *playbackProfile*, *hotkeyProfile*, *userIdentity*, *oneTimeKey*, *phoneticName*)

Connects to a server and displays it as tab in the client.

Parameters

`connectTab:` defines, which tab will be used, see ts3defines.PluginConnectTab
(type=int)

`serverLabel:` the label of the connection
(type=string)

`serverAddress:` host or ip address
(type=string)

`serverPassword:` password to the server, pass an empty string if the server is not password protected
(type=string)

`nickname:` the user's nickname
(type=string)

`channel:` complete path to the channel to connect to
(type=string)

`channelPassword:` password to the channel, pass an empty string if the channel is not password protected
(type=string)

`captureProfile:` the name of the capture profile to use
(type=string)

`playbackProfile:` the name of the playback profile to use
(type=string)

`hotkeyProfile:` the name of the hotkey profile to use
(type=string)

`userIdentity:` identity to use
(type=string)

`oneTimeKey:` privilege key to use on connect
(type=string)

`phoneticName:` phonetic nickname
(type=string)

Return Value

a tuple, containing the errorcode and the ID of the created serverconnection handler
(type=tuple (int, int))

guiConnectBookmark(*connectTab*, *bookmarkuuid*)

Connects to a server from a bookmark and displays it as tab in the client.

Parameters

`connectTab:` defines, which tab will be used, see ts3defines.PluginConnectTab

(type=int)

`bookmarkuuid:` UID of the bookmark

(type=string)

Return Value

a tuple, containing the errorcode and the ID of the created serverconnection handler

(type=tuple (int, int))

haltTransfer(*serverConnectionHandlerID*, *transferID*, *deleteUnfinishedFile*, *returnCode*)

Halts a currently running filetransfer.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection

(type=int)

`transferID:` the ID of the filetransfer

(type=int)

`deleteUnfinishedFile:` if set to 1 (or True) and the file is not yet finished, it will be deleted; to prevent, pass 0 (or False)

(type=int or bool)

`returnCode:` returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional.

(type=string)

Return Value

the errorcode

(type=int)

initiateGracefulPlaybackShutdown(*serverConnectionHandlerID*)

Graceful shutdown the playback device on a serverconnection.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

Return Value

the errorcode
(type=int)

isReceivingWhisper(*serverConnectionHandlerID*, *clientID*)

//FIXME:

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the ID of the client
(type=int)

Return Value

a tuple, containing the errorcode and the status
(type=tuple (int, int or bool))

isTransferSender(*transferID*)

Returns the upload/download direction of a filetransfer.

Parameters

`transferID:` the ID of the filetransfer
(type=int)

Return Value

a tuple, containing the errorcode and 1 if it's an upload or 0 if it's a download
(type=tuple (int, int or bool))

isWhispering(*serverConnectionHandlerID*, *clientID*)

Returns the status of a client whether he is currently whispering to the own client.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection

(type=int)

Return Value

a tuple, containing the errorcode and the status

(type=tuple (int, int or bool))

logMessage(*logMessage*, *severity*, *channel*, *logID*)

Logs a string.

Parameters

`logMessage:` Text which should be logged

(type=string)

`severity:` The level of the message, warning or error. Defined by the class LogLevel

(type=int)

`channel:` Custom text to categorize the message channel

(type=string)

`logID:` ID of the serverconnection to identify the current server connection when using multiple connections, 0 if unused

(type=int)

Return Value

The errorcode

(type=int)

openCaptureDevice(*serverConnectionHandlerID*, *modeID*, *captureDevice*)

Opens a playback device on a serverconnection.

Parameters

`serverConnectionHandlerID:`	ID of the serverconnection on which the capture device should be initialized on *(type=int)*
`modeID:`	the playback mode to use *(type=string)*
`captureDevice:`	the id of the capture device *(type=string)*

Return Value

the errorcode

(type=int)

openPlaybackDevice(*serverConnectionHandlerID*, *modeID*, *playbackDevice*)

Opens a playback device on a serverconnection.

Parameters

`serverConnectionHandlerID:`	ID of the serverconnection on which the playback device should be initialized on *(type=int)*
`modeID:`	the playback mode to use *(type=string)*
`playbackDevice:`	the id of the playback device *(type=string)*

Return Value

the errorcode

(type=int)

pauseWaveFileHandle(*serverConnectionHandlerID*, *waveHandle*, *pause*)

Pauses a wavefile sound previously started with playWaveFileHandle.

Parameters

`serverConnectionHandlerID:`	the ID of the serverConnection the sound is played on *(type=int)*
`waveHandle:`	the handle returned by playWaveFileHandle *(type=int)*
`pause:`	if set to 1 (or True), the sound will pause, 0 (or False) will unpause the sound *(type=int or bool)*

Return Value

the errorcode

(type=int)

playWaveFile(*serverConnectionHandlerID*, *path*)

Plays a wavefile sound on a serverconnection.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`path:`	the path to the wavefile on the system *(type=string)*

Return Value

the errorcode

(type=int)

playWaveFileHandle(*serverConnectionHandlerID*, *path*, *loop*)

Plays a wavefile sound on a serverconnection and returns a handle to it.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection on which the sound will be played on

(type=int)

`path:` the path to the wavefile on the system

(type=string)

`loop:` if set to 1 (or True), the sound will loop

(type=int or bool)

Return Value

A tuple, containing the errorcode and the handle, with which the sound can be paused and unpaused

(type=tuple (int, int))

printMessage(*serverConnectionHandlerID*, *message*, *messageTarget*)

Prints a message to a specific client chat tab.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection

(type=int)

`message:` the message to print

(type=string)

`messageTarget:` the target to send the message, see ts3defines.PluginMessageTarget

(type=int)

printMessageToCurrentTab(*message*)

Prints a message to the currently visible tab.

Parameters

`message:` the message to send

(type=string)

privilegeKeyUse(*serverConnectionHandlerID*, *tokenKey*, *returnCode*)

Uses a privilege key as the current client of the serverconnection.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`tokenKey:` the token
(type=string)

`returnCode:` returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional.
(type=string)

Return Value

the errorcode
(type=int)

processCustomCaptureData(*deviceName*, *buffer*)

Sends captured data from a custom device to the client libg

Parameters

`deviceName:` the name of the device capturing the data, previously registered with registerCustomDevice
(type=string)

`buffer:` a list containing the buffered data
(type=list [int])

Return Value

the errorcode
(type=int)

registerCustomDevice(*deviceID*, *deviceDisplayName*, *capFrequency*, *capChannels*, *playFrequency*, *playChannels*)

Registers a custom device, announcing the device ID and name to the Client Lib.

Parameters

`deviceID:`	ID string of the custom device, under which the device can be later accessed *(type=string)*
`deviceDisplayName:`	Displayed name of the custom device. Freely choose a name which identifies your device *(type=string)*
`capFrequency:`	Frequency of the capture device *(type=int)*
`capChannels:`	Number of channels of the capture device. This value depends on if the used codec is a mono or stereo CodecEncryptionMode *(type=int)*
`playFrequency:`	Frequency of the playback deviceDisplayName *(type=int)*
`playChannels:`	Number of channels of the playback device *(type=int)*

Return Value

the errorcode

(type=int)

requestBanList(*serverConnectionHandlerID*, *returnCode*)

Requests the banlist on a server. The event onBanListEvent will be triggered.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`returnCode:` returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional.
(type=string)

Return Value

the errorcode
(type=int)

requestChannelAddPerm(*serverConnectionHandlerID*, *channelID*, *permissionIDArray*, *permissionValueArray*, *returnCode*)

Adds a list of permissions to a channel.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelID:` the ID of the channel
(type=int)

`permissionIDArray:` a list of permission IDs
(type=list [int])

`permissionValueArray:` list of permission values, in order of the permissions in permissionIDArray
(type=list [int])

`returnCode:` returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional.
(type=string)

Return Value

the errorcode
(type=int)

requestChannelClientAddPerm(*serverConnectionHandlerID*, *channelID*, *clientDatabaseID*, *permissionIDArray*, *permissionValueArray*, *returnCode*)

Adds a list of permissions on a channel to a user.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel *(type=int)*
`clientDatabaseID:`	the database ID of the user *(type=int)*
`permissionIDArray:`	a list of permission IDs *(type=list [int])*
`permissionValueArray:`	list of permission values, in order of the permissions in permissionIDArray *(type=list [int])*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelClientDelPerm(*serverConnectionHandlerID*, *channelID*, *clientDatabaseID*, *permissionIDArray*, *returnCode*)

Deletes a list of permissions of a user in a channel.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel *(type=int)*
`clientDatabaseID:`	the database ID of the user *(type=int)*
`permissionIDArray:`	a list of permission IDs *(type=list [int])*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelClientPermList(*serverConnectionHandlerID*, *channelID*, *clientDatabaseID*, *returnCode*)

Requests the list of permissions of a user in a channel.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel *(type=int)*
`clientDatabaseID:`	the database ID of the user *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelDelete(*serverConnectionHandlerID*, *channelID*, *force*, *returnCode*)

Deletes a channel.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel to delete *(type=int)*
`force:`	if set to 1 (or True), the channel will be deleted even when it is not empty *(type=int or bool)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelDelPerm(*serverConnectionHandlerID*, *channelID*, *permissionIDArray*, *returnCode*)

Deletes a list of permissions from a channel.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel *(type=int)*
`permissionIDArray:`	a list of permission IDs *(type=list [int])*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelDescription(*serverConnectionHandlerID*, *channelID*, *returnCode*)

Requests the channel description of a channel. Afterwards, getChannelVariableAsString can return it.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelGroupAdd(*serverConnectionHandlerID*, *groupName*, *groupType*, *returnCode*)

Adds a channelgroup.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`groupName:`	the name of the channelgroup to create *(type=string)*
`groupType:`	type of the channelgroup, see ts3defines.GroupType *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelGroupAddPerm(*serverConnectionHandlerID*, *channelGroupID*, *continueonerror*, *permissionIDArray*, *permissionValueArray*, *returnCode*)

Adds a list of permissions to a channelgroup.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelGroupID:`	the ID of the channelgroup *(type=int)*
`continueonerror:`	if set to True, if an error with a permission occurs, the other permissions will even though be handled *(type=bool)*
`permissionIDArray:`	a list of permission IDs *(type=list [int])*
`permissionValueArray:`	list of permission values, in order of the permissions in permissionIDArray *(type=list [int])*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelGroupDel(*serverConnectionHandlerID*, *channelGroupID*, *force*, *returnCode*)

Deletes a channelgroup.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelGroupID:`	the ID of the channelgroup *(type=int)*
`force:`	if set to 1 (or True), even if there are users assigned to this channelgroup, it will be deleted *(type=int or bool)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelGroupDelPerm(*serverConnectionHandlerID*, *channelGroupID*, *continueOnError*, *permissionIDArray*, *returnCode*)

Deletes a list of permissions from a channelgroup.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelGroupID:` the ID of the channelgroup
(type=int)

`continueOnError:` if set to 1 (or True), if an error with a permission occurs, the other permissions will even though be handled
(type=int or bool)

`permissionIDArray:` a list of permission IDs
(type=list [int])

`returnCode:` returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional.
(type=string)

Return Value

the errorcode
(type=int)

requestChannelGroupList(*serverConnectionHandlerID*, *returnCode*)

Requests the list of channelgroups. The events onChannelGroupListEvent and onChannelGroupListEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelGroupPermList(*serverConnectionHandlerID*, *channelGroupID*, *returnCode*)

Requests the list of permissions assigned to a channelgroup. The events onChannelGroupPermListEvent and onChannelGroupPermListFinishedEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelGroupID:`	the ID of the channelGroupID *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelMove(*serverConnectionHandlerID*, *channelID*, *newChannelParentID*, *newChannelOrder*, *returnCode*)

Moves a channel to a new parent channel.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel to move *(type=int)*
`newChannelParentID:`	the ID of the new parent channel *(type=int)*
`newChannelOrder:`	Channel order defining where the channel should be sorted under the new parent. Pass 0 to sort the channel right after the parent *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelPermList(*serverConnectionHandlerID*, *channelID*, *returnCode*)

Requests the list of permissions assigned to a channel. The events onChannelPermListEvent and onChannelPermListFinishedEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelSubscribe(*serverConnectionHandlerID*, *channelIDArray*, *returnCode*)

Subscribes to a list of channels to get notifications of the clients in them.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelIDArray:`	a list of channel IDs *(type=list [int])*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelSubscribeAll(*serverConnectionHandlerID*, *returnCode*)

Subscribes to all channels on the server.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelUnsubscribe(*serverConnectionHandlerID*, *channelIDArray*, *returnCode*)

Unsubscribes from a list channels.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelIDArray:`	a list of channel IDs *(type=list [int])*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestChannelUnsubscribeAll(*serverConnectionHandlerID*, *returnCode*)

Unsubscribes from all channels on the server.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestClientAddPerm(*serverConnectionHandlerID*, *clientDatabaseID*, *permissionIDArray*, *permissionValueArray*, *permissionSkipArray*, *returnCode*)

Adds a list of permissions to a user.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientDatabaseID:`	the database ID of the user *(type=int)*
`permissionIDArray:`	a list of permission IDs *(type=list [int])*
`permissionValueArray:`	list of permission values, in order of the permissions in permissionIDArray *(type=list [int])*
`permissionSkipArray:`	list of permission skip values, in order of the permissions in permissionIDArray *(type=list [int])*

Return Value

the errorcode

(type=int)

requestClientDBIDfromUID(*serverConnectionHandlerID*, *clientUniqueIdentifier*, *returnCode*)

Requests the database ID of a client defined by the UID. The event onClientDBIDfromUIDEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientUniqueIdentifier:`	the UID of the client *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestClientDelPerm(*serverConnectionHandlerID*, *clientDatabaseID*, *permissionIDArray*, *returnCode*)

Deletes a list of permissions from a user.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientDatabaseID:`	the database ID of the user *(type=int)*
`permissionIDArray:`	a list of permission IDs *(type=list [int])*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestClientEditDescription(*serverConnectionHandlerID*, *clientID*, *clientDescription*, *returnCode*)

Sets the description of a client.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the ID of the client *(type=int)*
`clientDescription:`	the description to set *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestClientIDs(*serverConnectionHandlerID*, *clientUniqueIdentifier*, *returnCode*)

Requests the client IDs for a given UID. Will trigger the event onClientIDsEvent.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientUniqueIdentifier:`	the UID of the client *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestClientKickFromChannel(*serverConnectionHandlerID*, *clientID*, *kickReason*, *returnCode*)

Kicks a client from its current channel to the default one.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the ID of the client to kick *(type=int)*
`kickReason:`	the reason for the kick *(type=string)*

Return Value

the errorcode

(type=int)

requestClientKickFromServer(*serverConnectionHandlerID*, *clientID*, *kickReason*, *returnCode*)

Kicks a client from the server.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the ID of the client to kick *(type=int)*
`kickReason:`	the reason for the kick *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestClientMove(*serverConnectionHandlerID*, *clientID*, *newChannelID*, *password*, *returnCode*)

Moves a client to a different channel.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the ID of the client to be moved *(type=int)*
`newChannelID:`	the ID of the channel moving the client to *(type=int)*
`password:`	password of the channel, leave empty if channel is not password protected *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestClientNamefromDBID(*serverConnectionHandlerID*, *clientDatabaseID*, *returnCode*)

Requests the name of a client defined by the database ID. The event onClientNamefromDBIDEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientDatabaseID:`	the database ID of the client *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestClientNamefromUID(*serverConnectionHandlerID*, *clientUniqueIdentifier*, *returnCode*)

Requests the name of a client defined by the UID. The event onClientNamefromUIDEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientUniqueIdentifier:`	the UID of the client *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestClientPermList(*serverConnectionHandlerID*, *clientDatabaseID*, *returnCode*)

Requests the list of permissions assigned to a user. The events

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientDatabaseID:`	the database ID of the user *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestClientPoke(*serverConnectionHandlerID*, *clientID*, *message*, *returnCode*)

Pokes a client with a given message.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the ID of the client *(type=int)*
`message:`	the message *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestClientSetIsTalker(*serverConnectionHandlerID*, *clientID*, *isTalker*, *returnCode*)

Grants or revokes the talker flag of a client.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the ID of the client *(type=int)*
`isTalker:`	if set to 1 (or True) grants talker flag, if 0 (or False) revokes talker flag *(type=int or bool)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestClientSetWhisperList(*serverConnectionHandlerID*, *clientID*, *targetChannelIDArray*, *targetClientIDArray*, *returnCode*)

Modifies the whisper list of a client.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the ID of the client whose whisperlist is modified. If set to 0, the own whisper list is modified *(type=int)*
`targetChannelIDArray:`	a list of channel IDs the client will whisper to *(type=list [int])*
`targetClientIDArray:`	a list of client IDs the client will whisper to *(type=list [int])*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestClientVariables(*serverConnectionHandlerID*, *clientID*, *returnCode*)

Requests latest data for a given client. The event onUpdateClientEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the ID of the client *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestComplainAdd(*serverConnectionHandlerID*, *targetClientDatabaseID*, *complainReason*, *returnCode*)

Adds a complain to a user defined by his database ID.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`targetClientDatabaseID:`	the database ID of the user *(type=int)*
`complainReason:`	the reason for the complain *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestComplainDel(*serverConnectionHandlerID*, *targetClientDatabaseID*, *fromClientDatabaseID*, *returnCode*)

Deletes a complain to a user by a different user.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`targetClientDatabaseID:`	the database ID of the complained user *(type=int)*
`fromClientDatabaseID:`	the database ID of the complaining user *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestComplainDelAll(*serverConnectionHandlerID*, *targetClientDatabaseID*, *returnCode*)

Deletes all complains to a user.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`targetClientDatabaseID:`	the database ID of the user *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestComplainList(*serverConnectionHandlerID*, *targetClientDatabaseID*, *returnCode*)

Requests the list of complains to a user. The event onComplainListEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`targetClientDatabaseID:`	the database ID of the user *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestConnectionInfo(*serverConnectionHandlerID*, *clientID*, *returnCode*)

Requests the connection info of a client. The event onConnectionInfoEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the ID of the client *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestCreateDirectory(*serverConnectionHandlerID*, *channelID*, *channelPW*, *directoryPath*, *returnCode*)

Creates a directory in a channel.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel *(type=int)*
`channelPW:`	the password of the channel, pass an empty string if channel is not password protected *(type=string)*
`directoryPath:`	the complete path of the to be created directory *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestDeleteFile(*serverConnectionHandlerID*, *channelID*, *channelPW*, *files*, *returnCode*)

Deletes a list of files in a channel.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel *(type=int)*
`channelPW:`	the password of the channel, pass an empty string if channel is not password protected *(type=string)*
`files:`	a list of complete pathes of the file to delete *(type=list [string])*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestFile(*serverConnectionHandlerID*, *channelID*, *channelPW*, *file*, *overwrite*, *resume*, *destinationDirectory*, *returnCode*)

Starts a filedownload from the server.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel in which the file is placed in *(type=int)*
`channelPW:`	the password of the channel, pass an empty string if the channel is not password protected *(type=string)*
`file:`	the complete path to the file in the channel *(type=string)*
`overwrite:`	if set to 1 (or True) and a file with that name already exists will be overwritten *(type=int or bool)*
`resume:`	if set to 1 (or True), a previously started filetransfer can be resumed *(type=int or bool)*
`destinationDirectory:`	the path to the directory, where the downloaded fill will be placed in *(type=strin)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

a tuple, containing the errorcode and the ID of the filetransfer

(type=tuple (int, int))

requestFileInfo(*serverConnectionHandlerID*, *channelID*, *channelPW*, *file*, *returnCode*)

Requests the info to a file in a channel. The event onFileInfoEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel *(type=int)*
`channelPW:`	the password of the channel, pass an empty string if the channel is not password protected *(type=string)*
`file:`	the complete path to the file *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestFileList(*serverConnectionHandlerID*, *channelID*, *channelPW*, *path*, *returnCode*)

Requests the filelist of a channel. The events onFileListEvent and onFileListFinishedEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel *(type=int)*
`channelPW:`	the password of the channel, pass an empty string if the channel is not password protected *(type=string)*
`path:`	the path of the directory to be listed, pass '/' for the root path *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestHotkeyInputDialog(*keyword*, *isDown*, *qParentWindow*)

Shows the hotkeyinputdialog to set the hotkey for a hotkey keyword. When finished ts3plugin.onHotkeyRecordedEvent will be called with the recorded hotkey.

Parameters

`keyword:` the global keyword (see PluginHost.globalHotkeyKeyword)

(type=str)

`isDown:` if True, the hotkey will be triggered on keypress, on keyrelease otherwise

(type=bool)

`qParentWindow:` the window on which the dialog is shown modal to, optional

(type=QWidget)

requestInfoUpdate(*serverConnectionHandlerID*, *itemType*, *itemID*)

Requests to update the info data.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection

(type=int)

`itemType:` specifies, which info data update is requested, see ts3defines.PluginItemType

(type=int)

`itemID:` the ID of the item to update (only usefull if itemType != ts3defines.PluginItemType.PLUGIN_MENU_TYPE_GLOBAL)

(type=int)

Return Value

the errorcode

(type=int)

requestIsTalker(*serverConnectionHandlerID*, *isTalkerRequest*, *isTalkerRequestMessage*, *returnCode*)

Requests talk power or revokes the talk power request.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`isTalkerRequest:`	if set to 1 (or True) requests talk power, if 0 (or False) revokes the talk power request *(type=int or bool)*
`isTalkerRequestMessage:`	the message of the request *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestMessageAdd(*serverConnectionHandlerID*, *toClientUID*, *subject*, *message*, *returnCode*)

Sends an offline message to another user.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`toClientUID:`	the UID of the user *(type=string)*
`subject:`	the subject of the message *(type=string)*
`message:`	the message *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestMessageDel(*serverConnectionHandlerID*, *messageID*, *returnCode*)

Deletes an offline message.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`messageID:`	the ID of the message *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestMessageGet(*serverConnectionHandlerID*, *messageID*, *returnCode*)

Requests an offline message defined by its ID. The event onMessageGetEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`messageID:`	the ID of the message *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestMessageList(*serverConnectionHandlerID*, *returnCode*)

Requests the list of offline messages. The event onMessageListEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestMessageUpdateFlag(*serverConnectionHandlerID*, *messageID*, *flag*, *returnCode*)

Sets the message read/unread flag of an offline message

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`messageID:`	the ID of the message *(type=int)*
`flag:`	set to 0 to set message as unread, set to 1 to set message as read *(type=)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestMuteClients(*serverConnectionHandlerID*, *clientIDArray*, *returnCode*)

Mutes a list of clients.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientIDArray:`	a list of client IDs *(type=list [int])*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestPermissionList(*serverConnectionHandlerID*, *returnCode*)

Requests the list of permissions available on the server. The events onPermissionListEvent and onPermissionListFinishedEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestPermissionOverview(*serverConnectionHandlerID*, *clientDBID*, *channelID*, *returnCode*)

Requests the permission overview of a user in a channel. The events onPermissionOverviewEvent and onPermissionOverviewFinishedEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientDBID:`	the database ID of the user *(type=int)*
`channelID:`	the ID of the channel *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestRenameFile(*serverConnectionHandlerID*, *fromChannelID*, *channelPW*, *toChannelID*, *toChannelPW*, *oldFile*, *newFile*, *returnCode*)

Renames a file or moves it to another channel.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`fromChannelID:`	the ID of the channel, the file is currently placed in *(type=int)*
`channelPW:`	the password of the channel, the file is currently placed in, pass an empty string if channel is not password protected *(type=string)*
`toChannelID:`	the ID of the channel, the file should be placed in after, pass 0, if just renaming, not moving *(type=int)*
`toChannelPW:`	the password of the channel, to which the file should move to, pass an empty string if channel is not password protected; this is ignored, if just renaming, not moving *(type=string)*
`oldFile:`	the complete path to the file *(type=string)*
`newFile:`	the complete path to the new filename *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestSendChannelTextMsg(*serverConnectionHandlerID*, *message*, *targetChannelID*, *returnCode*)

Sends a text message to all clients in a channel.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`message:`	the message to send *(type=string)*
`targetChannelID:`	the ID of the channel *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestSendClientQueryCommand(*serverConnectionHandlerID*, *command*, *returnCode*)

Requests to execute a clientquery command.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`command:`	the command to execute *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestSendPrivateTextMsg(*serverConnectionHandlerID*, *message*, *targetClientID*, *returnCode*)

Sends a private text message to a client.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`message:`	the message to send *(type=string)*
`targetClientID:`	the ID of the client to send the message to *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestSendServerTextMsg(*serverConnectionHandlerID*, *message*, *returnCode*)

Sends a text message to all clients on the server.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`message:`	the message to send *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestServerGroupAdd(*serverConnectionHandlerID*, *groupName*, *groupType*, *returnCode*)

Adds a servergroup.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`groupName:`	the name of the group to create *(type=string)*
`groupType:`	type of the servergroup, see ts3defines.GroupType *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestServerGroupAddClient(*serverConnectionHandlerID*, *serverGroupID*, *clientDatabaseID*, *returnCode*)

Adds a user defined by his database ID to a servergroup.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`serverGroupID:`	the ID of the servergroup *(type=int)*
`clientDatabaseID:`	the database ID of the user *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestServerGroupAddPerm(*serverConnectionHandlerID*, *serverGroupID*, *continueonerror*, *permissionIDArray*, *permissionValueArray*, *permissionNegatedArray*, *permissionSkipArray*, *returnCode*)

Adds a list of permissions to a servergroup.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`serverGroupID:`	the ID of the servergroup *(type=int)*
`continueonerror:`	if set to 1 (or True), if an error with a permission occurs, the other permissions will even though be handled *(type=int or bool)*
`permissionIDArray:`	list of permission IDs *(type=list [int])*
`permissionValueArray:`	list of permission values, in order of the permissions in permissionIDArray *(type=list [int])*
`permissionNegatedArray:`	list of permission negated values, in order of the permissions in permissionIDArray *(type=list [int])*
`permissionSkipArray:`	list of permission skip values, in order of the permissions in permissionIDArray *(type=list [int])*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestServerGroupClientList(*serverConnectionHandlerID*, *serverGroupID*, *withNames*, *returnCode*)

Requests the list of clients assigned to a servergroup. The event onServerGroupClientListEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`serverGroupID:`	the ID of the servergroup *(type=int)*
`withNames:`	if set to 1 (or True), the event will contain the nick and uid of the user instead of empty strings *(type=int or bool)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestServerGroupDel(*serverConnectionHandlerID*, *serverGroupID*, *force*, *returnCode*)

Deletes a servergroup.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`serverGroupID:`	the ID of the servergroup *(type=int)*
`force:`	if set to 1 (or True), even if there are users assigned to this servergroup, it will be deleted *(type=int or bool)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestServerGroupDelClient(*serverConnectionHandlerID*, *serverGroupID*, *clientDatabaseID*, *returnCode*)

Deletes a user defined by his database ID from a servergroup.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`serverGroupID:`	the ID of the servergroup *(type=int)*
`clientDatabaseID:`	the database ID of the user *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestServerGroupDelPerm(*serverConnectionHandlerID*, *serverGroupID*, *continueOnError*, *permissionIDArray*, *returnCode*)

Deletes a list of permissions from a servergroup.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`serverGroupID:`	the ID of the servergroup *(type=int)*
`continueOnError:`	if set to 1 (or True), if an error with a permission occurs, the other permissions will even though be handled *(type=int or bool)*
`permissionIDArray:`	list of permission IDs *(type=list [int])*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestServerGroupList(*serverConnectionHandlerID*, *returnCode*)

Requests the list of servergroups. The events onServerGroupListEvent and onServerGroupListFinishedEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestServerGroupPermList(*serverConnectionHandlerID*, *serverGroupID*, *returnCode*)

Requests the list of permissions assigned to a servergroup. The events onServerGroupPermListEvent and onServerGroupPermListFinishedEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`serverGroupID:`	the ID of the servergroup *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestServerGroupsByClientID(*serverConnectionHandlerID*, *clientDatabaseID*, *returnCode*)

Requests all servergroups of a user defined by his database ID. The event onServerGroupByClientIDEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientDatabaseID:`	the database ID of the user *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestServerTemporaryPasswordAdd(*serverConnectionHandlerID*, *password*, *description*, *duration*, *targetChannelID*, *targetChannelPW*, *returnCode*)

Adds a temporary password to the server.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`password:`	the temporary password *(type=string)*
`description:`	the description of the temporary password *(type=string)*
`duration:`	the duration in seconds *(type=int)*
`targetChannelID:`	the ID of the channel to which the accessing clients will join by default *(type=int)*
`targetChannelPW:`	the password of the targetChannel, pass an empty string, if the channel is not password protected *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestServerTemporaryPasswordDel(*serverConnectionHandlerID*, *password*, *returnCode*)

Deletes an existing temporary password.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`password:`	the password to delete *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestServerTemporaryPasswordList(*serverConnectionHandlerID*, *returnCode*)

Requests a list of existing temporary passwords. The event onServerTemporaryPasswordListEvent will be triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

requestServerVariables(*serverConnectionHandlerID*)

Requests all server variables of a serverconnection. The event onServerUpdatedEvent will be triggered.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

Return Value

the errorcode
(type=int)

requestSetClientChannelGroup(*serverConnectionHandlerID*, *channelGroupIDArray*, *channelIDArray*, *clientDatabaseIDArray*, *returnCode*)

Adds a list of users to a list of channelgroups.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelGroupIDArray:` a list of channelgroup IDs
(type=list [int])

`channelIDArray:` a list of channel IDs
(type=list [int])

`clientDatabaseIDArray:` a list of client database IDs
(type=list [int])

`returnCode:` returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional.
(type=string)

Return Value

the errorcode
(type=int)

requestUnmuteClients(*serverConnectionHandlerID*, *clientIDArray*, *returnCode*)

Unmutes a list of clients.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientIDArray:`	a list of client IDs *(type=list [int])*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

sendFile(*serverConnectionHandlerID*, *channelID*, *channelPW*, *file*, *overwrite*, *resume*, *sourceDirectory*, *returnCode*)

Starts a fileupload to the server.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel in which the file will be placed in *(type=int)*
`channelPW:`	the password of the channel, pass an empty string if the channel is not password protected *(type=string)*
`file:`	the complete path to the file in the channel *(type=string)*
`overwrite:`	if set to 1 (or True) and a file with that name already exists will be overwritten *(type=int or bool)*
`resume:`	if set to 1 (or True), a previously started filetransfer can be resumed *(type=int or bool)*
`sourceDirectory:`	the directory on the system, where the original file is placed in *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

a tuple, containing the errorcode and the ID of the filetransfer

(type=tuple (int, int))

sendPluginCommand(*serverConnectionHandlerID*, *command*, *targetMode*, *targetIDs*, *returnCode*)

Sends a plugin command to other users.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`command:` the command string
(type=string)

`targetMode:` specifies, to whom the command will be send, see ts3defines.PluginTargetMode
(type=int)

`targetIDs:` a list of client IDs, only needed if targetMode == ts3defines.PluginTargetMode.PluginCommandTarget_CLIENT
(type=list [int])

serverPropertyStringToFlag(*serverPropertyString*)

Converts a server property name used in strings (eg the serverquery) to the corresponding flag.

Parameters

`serverPropertyString:` the lowercase string representation
(type=str)

Return Value

a tuple, containing the errorcode and the flag (see ts3defines.ClientProperties and ts3defines.ClientPropertiesRare)
(type=tuple (int, int))

set3DWaveAttributes(*serverConnectionHandlerID*, *waveHandle*, *position*)

Positions a wave file that was opened previously with playWaveFileHandle in 3D space.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`waveHandle:` handle of the played wavefile sound returned by playWaveFileHandle
(type=int)

`position:` A tuple defining the 3D position of the sound
(type=tuple (float, float, float))

Return Value

the errorcode
(type=int)

setChannelVariableAsInt(*serverConnectionHandlerID*, *channelID*, *flag*, *value*)

Sets a channel variable to a new int value.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelID:` the ID of the channel, pass 0 to set a new channel's variables
(type=int)

`flag:` the flag to set
(type=int)

`value:` the new value
(type=int)

Return Value

the errorcode
(type=int)

setChannelVariableAsString(*serverConnectionHandlerID*, *channelID*, *flag*, *value*)

Sets a channel variable to a new string value.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel, pass 0 to set a new channel's variables *(type=int)*
`flag:`	the flag to set *(type=int)*
`value:`	the new value *(type=int)*

Return Value

the errorcode

(type=int)

setChannelVariableAsUInt64(*serverConnectionHandlerID*, *channelID*, *flag*, *value*)

Sets a channel variable to a new unsigned long long int value.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the ID of the channel, pass 0 to set a new channel's variables *(type=int)*
`flag:`	the flag to set *(type=int)*
`value:`	the new value *(type=int)*

Return Value

the errorcode

(type=int)

setClientSelfVariableAsInt(*serverConnectionHandlerID*, *flag*, *value*)

Sets a variable of the own client to a new int value.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`flag:`	the flag to set *(type=int)*
`value:`	the new value *(type=int)*

Return Value

the errorcode

(type=int)

setClientSelfVariableAsString(*serverConnectionHandlerID*, *flag*, *value*)

Sets a variable of the own client to a new string value.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`flag:`	the flag to set *(type=int)*
`value:`	the new value *(type=string)*

Return Value

the errorcode

(type=int)

setClientVolumeModifier(*serverConnectionHandlerID*, *clientID*, *value*)

Sets the volume modifier of a client.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the client's ID *(type=int)*
`value:`	the value to set *(type=float)*

Return Value

the errorcode

(type=int)

setPlaybackConfigValue(*serverConnectionHandlerID*, *ident*, *value*)

Sets a playback option.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`ident:`	the playback option to reset *(type=string)*
`value:`	the value to set *(type=string)*

Return Value

the errorcode

(type=int)

setPluginMenuEnabled(*menuID*, *enabled*)

Enables or disables a menuitem. The menuID must be the global id, not the local id plugin developers set in menuItems. Retrieve it with PluginHost.globalMenuID.

Parameters

`menuID:`	global id of the menuitem *(type=int)*
`enabled:`	set to True to enable it, False otherwise *(type=bool)*

setPreProcessorConfigValue(*serverConnectionHandlerID*, *ident*, *value*)

Sets a sound preprocessor flag.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`ident:` the flag to be set
(type=string)

`value:` the value to set the flag to
(type=string)

Return Value

the errorcode
(type=int)

showHotkeySetup()

Opens the hotkey settings in the TeamSpeak 3 client's settings dialog.

spawnNewServerConnectionHandler(*port*)

Creates a new server connection handler and receive its ID.

Parameters

`port:` Port the client should bind on. Specify zero to let the operating system chose any free port
(type=int)

Return Value

A tuple, containig the errorcode and the resulting ID
(type=tuple (int, int))

startConnection(*serverConnectionHandlerID*, *identity*, *ip*, *port*, *nickname*, *defaultChannelArray*, *defaultChannelPassword*, *serverPassword*)

Starts a connection to the given server.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`identity:`	the client's identity *(type=string)*
`ip:`	hostname or ip of the server *(type=string)*
`port:`	port of the server *(type=int)*
`nickname:`	the client's nickname *(type=string)*
`defaultChannelArray:`	list of strings defining the path to a channel on the server, pass empty list to join in server's default channel *(type=list [string])*
`defaultChannelPassword:`	password of the default channel, pass an empty string if not using defaultChannelArray or channel is not password protected *(type=string)*
`serverPassword:`	password of the server, pass an empty string if the server is not password protected *(type=string)*

Return Value

the errorcode

(type=int)

startVoiceRecording(*serverConnectionHandlerID*)

Starts voice recording on a serverconnection.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

Return Value

the errorcode
(type=int)

stopConnection(*serverConnectionHandlerID*, *quitMessage*)

Stops the connection of a serverconnection.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`quitMessage:` a message displayed when leaving the server encoded in UTF-8
(type=string)

Return Value

the errorcode
(type=int)

stopVoiceRecording(*serverConnectionHandlerID*)

Stops voice recording on a serverconnection

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

Return Value

the errorcode
(type=int)

systemset3DListenerAttributes(*serverConnectionHandlerID*, *position*, *forward*, *up*)

Sets the position, velocity and orientation of the own client in 3D space

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection

(type=int)

`position:` A tuple defining the 3D position, pass None to ignore

(type=tuple (float, float, float))

`forward:` A tuple defining the forward orientation of the listener. The vector must be of unit length and perpendicular to the up vector. Pass None to ignore.

(type=tuple (float, float, float))

`up:` A tuple defining the upward orientation of the listener. The vector must be of unit length and perpendicular to the forward vector. Pass None to ignore.

(type=tuple (float, float, float))

Return Value

the errorcode

(type=int)

systemset3DSettings(*serverConnectionHandlerID*, *distanceFactor*, *rolloffScale*)

Adjust 3D sound system settings.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`distanceFactor:`	relative distance factor. Default is 1.0 = 1 meter *(type=float)*
`rolloffScale:`	Scaling factor for 3D sound rolloff. Defines how fast sound volume will attenuate. As higher the value, as faster the sound is toned with increasing distance. *(type=float)*

Return Value

the errorcode

(type=int)

unregisterCustomDevice(*deviceID*)

Unregisters a custom device, previously registered with registerCustomDevice.

Parameters

`deviceID:`	the ID of the device, used in registerCustomDevice *(type=string)*

Return Value

the errorcode

(type=int)

urlsToBB(*text*, *maxLen*=256)

Converts an url to the BB-code respresentation.

Parameters

`text:` the url

(type=string)

`maxLen:` length of the buffer, passed to the clientlib to store the path to, default value is 256

(type=int)

Return Value

the BB-code representation

(type=string)

verifyChannelPassword(*serverConnectionHandlerID*, *channelID*, *channelPassword*, *returnCode*)

Verifies the password to a channel. The result can be checked in onServerErrorEvent.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection

(type=int)

`channelID:` the ID of the channel

(type=int)

`channelPassword:` the password to be verified

(type=string)

`returnCode:` returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional.

(type=string)

Return Value

the errorcode

(type=int)

verifyServerPassword(*serverConnectionHandlerID*, *serverPassword*, *returnCode*)

Verifies the password to a server. The result can be checked in onServerErrorEvent.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`serverPassword:`	the password to be verified *(type=string)*
`returnCode:`	returnCode passed to onServerErrorEvent or onServerPermissionErrorEvent. Optional. *(type=string)*

Return Value

the errorcode

(type=int)

14 Module ts3plugin

14.1 Class PluginMount

object
  type
    ts3plugin.PluginMount

Mountpoint for ts3plugins. This class is used as metaclass for ts3plugin-subclasses to autodetect classes and add them to the PluginHost.

14.1.1 Methods

__init__(*cls*, *name*, *bases*, *attrs*)

x.__init__(...) initializes x; see help(type(x)) for signature

Return Value
the object's type

Overrides: object.__init__ extit(inherited documentation)

Inherited from type

__call__(), __delattr__(), __eq__(), __ge__(), __getattribute__(), __gt__(), __hash__(), __instancecheck__(), __le__(), __lt__(), __ne__(), __new__(), __repr__(), __setattr__(), __subclasscheck__(), __subclasses__(), mro()

Inherited from object

__format__(), __reduce__(), __reduce_ex__(), __sizeof__(), __str__(), __subclasshook__()

14.1.2 Properties

Name	Description
Inherited from type __abstractmethods__, __base__, __bases__, __basicsize__, __dictoffset__, __flags__, __itemsize__, __mro__, __name__, __weakrefoffset__	
Inherited from object __class__	

14.2 Class ts3plugin

object
└ **ts3plugin.ts3plugin**

14.2.1 Methods

__init__(*self*)

Initializes the plugin. This is called if the plugin is started. After this, the plugin's event-methods will be invoked.

Overrides: object.__init__

stop(*self*)

This is called, when the plugin is stopped. After this, event-methods won't be invoked any longer.

menuCreated(*self*)

This is called after the plugin's menuitems are created or the plugin was reactivated. Plugin developers can assume, that when this is called, all menuitems are enabled, disable them with ts3lib.setPluginMenuEnabled if necessary.

configure(*self*, *qParentWidget*)

This is called to show the plugin's configuration ui.

Parameters

`qParentWidget`: the reference to pyTSon's configdialog
(type=pytsonui.ConfigurationDialog)

infoData(*self*, *schid*, *aid*, *atype*)

If the classvariable infoTitle is not None, this is called to show information on a treeitem of the TS3 Client.

Parameters

`schid`: the ID of the serverconnection

(type=int)

`aid`: the id (channel or client id) of the object represented by the treeitem

(type=int)

`atype`: type of the treeitem (see ts3defines.PluginItemType)

(type=int)

Return Value

list of strings shown in the client (will be joined by a newline)

(type=list[str])

processCommand(*self*, *schid*, *command*)

If the classvariable commandKeyword is set to a string (non-empty), this is called if the user requests a command by typing /py commandKeyword [args].

Parameters

`schid`: the ID of the serverconnection

(type=int)

`command`: the additional arguments passed by the user

(type=str)

Return Value

True, if the plugin handled the command, otherwise the user will receive an error

(type=bool)

onServerErrorEvent(*self*, *schid*, *errorMessage*, *error*, *returnCode*, *extraMessage*)

This is the global error event. Independent from the return value, all pyTSon plugins will receive this event.

Parameters

`schid:` the ID of the serverconnection

(type=int)

`errorMessage:` the message

(type=str)

`error:` the errorcode (see ts3defines.ERROR_*)

(type=int)

`returnCode:` the returnCode of the error passed to the causal method or an empty string, if no returnCode was passed

(type=str)

`extraMessage:` additional error information

(type=str)

Return Value

True, if the plugin handled the command, so the client will ignore it. If no returnCode was passed, this return value will be ignored

(type=bool)

onTextMessageEvent(*self*, *schid*, *targetMode*, *toID*, *fromID*, *fromName*, *fromUniqueIdentifier*, *message*, *ffIgnored*)

This is called when the client receives a textmessage from another client. Independent from the return value, all pyTSon plugins will receive this event.

Parameters

`schid:` the ID of the serverconnection

(type=int)

`targetMode:` the target of the message (see ts3defines.TextMessageTargetMode)

(type=int)

`toID:` the id of the receiver (client or channel)

(type=int)

`fromID:` the client id of the sending client

(type=int)

`fromName:` the current nick of the sending client

(type=str)

`fromUniqueIdentifier:` the uid of the sending client

(type=str)

`message:` the message

(type=str)

`ffIgnored:` if set to a value != 0, the client will ignore this message independent from the return value (eg. the friend/foe manager kicked in)

(type=int)

Return Value

True, if the plugin handled the message, so the client will ignore the message

(type=bool)

onClientPokeEvent(*self*, *schid*, *fromClientID*, *pokerName*, *pokerUniqueIdentity*, *message*, *ffIgnored*)

This is called when the client is poked by another client. Independent from the return value, all pyTSon plugins will receive this event.

Parameters

`schid:` the ID of the serverconnection
(type=int)

`fromClientID:` the id of the poking client
(type=int)

`pokerName:` the current nick of the poking client
(type=str)

`pokerUniqueIdentity:` the uid of the poking client
(type=str)

`message:` the poke message
(type=str)

`ffIgnored:` if set to a value != 0, the client will ignore this message independent from the return value (eg. the friend/foe manager kicked in)
(type=int)

Return Value

True, if the plugin handled the poke, so the client will ignore it
(type=bool)

onServerPermissionErrorEvent(*self*, *schid*, *errorMessage*, *error*, *returnCode*, *failedPermissionID*)

This is the global error event for permission errors. Independent from the return value, all pyTSon plugins will receive this event.

Parameters

`schid:` the ID of the serverconnection

(type=int)

`errorMessage:` the message

(type=str)

`error:` the errorcode (see ts3defines.ERROR_*)

(type=int)

`returnCode:` the returnCode of the error passed to the causal method or an empty string, if no returnCode was passed

(type=str)

`failedPermissionID:` id of the permission

(type=int)

Return Value

True, if the plugin handled the error, so the client will ignore it. If no returnCode was passed, this return value will be ignored

(type=bool)

onUserLoggingMessageEvent(*self*, *logMessage*, *logLevel*, *logChannel*, *logID*, *logTime*, *completeLogString*)

This is called whenever a message is added to the clientlog. You should not call ts3lib.logMessage in this event to prevent infinite loops. This event can be called asynchronous if called from another thread than the mainthread.

Parameters

`logMessage:`	the message that has been logged *(type=str)*
`logLevel:`	the level of the message (see ts3defines.LogLevel) *(type=int)*
`logChannel:`	the logchannel of the message *(type=str)*
`logID:`	the id of the server connection handler it the message is connected to one, otherwise set to 0 *(type=int)*
`logTime:`	the time of the message as unix timestamp *(type=int)*
`completeLogString:`	all infos concatenated as string *(type=str)*

onFileTransferStatusEvent(*self*, *transferID*, *status*, *statusMessage*, *remotefileSize*, *schid*)

This is called whenever a filetransfer's status changed. This event is called asynchronous.

Parameters

`transferID:`	the id of the filetransfer *(type=int)*
`status:`	the new status *(type=int)*
`statusMessage:`	a statusmessage *(type=str)*
`remotefileSize:`	size of the file on the remote site (if uploading, this is the incompletefilesize) *(type=int)*
`schid:`	the ID of the serverconnection *(type=int)*

currentServerConnectionChanged(*self*, *serverConnectionHandlerID*)

This is called when the current serverconnection changed (the user switched between tabs)

Parameters

`serverConnectionHandlerID:`	id of the new serverconnectionhandler *(type=int)*

onAvatarUpdated(*self*, *serverConnectionHandlerID*, *clientID*, *avatarPath*)

This is called when a client's avatar changed.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the client's id *(type=int)*
`avatarPath:`	path to the avatar *(type=str)*

onBanListEvent(*self*, *serverConnectionHandlerID*, *banid*, *ip*, *name*, *uid*, *creationTime*, *durationTime*, *invokerName*, *invokercldbid*, *invokeruid*, *reason*, *numberOfEnforcements*, *lastNickName*)

This is called for each entry in the server's banlist after it was requested with ts3lib.requestBanList.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`banid:` id of the ban
(type=int)

`ip:` the banned ip pattern or an empty string
(type=str)

`name:` the banned name pattern or an empty string
(type=str)

`uid:` the banned uid or an empty string
(type=str)

`creationTime:` time the ban was created as unix timestamp
(type=int)

`durationTime:` duration of the ban in seconds
(type=int)

`invokerName:` nick of the creator (at time the ban was created)
(type=str)

`invokercldbid:` database id of the creator
(type=int)

`invokeruid:` uid of the creator
(type=str)

`reason:` reason for ban
(type=str)

`numberOfEnforcements:` number of times, the ban has been enforced since
(type=int)

`lastNickName:` last nickname of the last enforced client
(type=str)

onChannelClientPermListEvent(*self*, *serverConnectionHandlerID*, *channelID*, *clientDatabaseID*, *permissionID*, *permissionValue*, *permissionNegated*, *permissionSkip*)

This is called for each granted permission of a client in a specific channel requested with ts3lib.requestChannelClientPermList.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	id of the channel *(type=int)*
`clientDatabaseID:`	the client's database id *(type=int)*
`permissionID:`	id of the permission *(type=int)*
`permissionValue:`	value of the permission *(type=int)*
`permissionNegated:`	the negated flag *(type=int)*
`permissionSkip:`	the skip flag *(type=int)*

onChannelClientPermListFinishedEvent(*self*, *serverConnectionHandlerID*, *channelID*, *clientDatabaseID*)

This is called after each permission yielded by onChannelClientPermListEvent was triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	id of the channel *(type=int)*
`clientDatabaseID:`	the client's database id *(type=int)*

onChannelDescriptionUpdateEvent(*self*, *serverConnectionHandlerID*, *channelID*)

This is called whenever a channel's description is updated.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelID:` id of the channel
(type=int)

onChannelGroupListEvent(*self*, *serverConnectionHandlerID*, *channelGroupID*, *name*, *atype*, *iconID*, *saveDB*)

This is called for each channelgroup on the server requested with ts3lib.requestChannelGroupList.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelGroupID:` id of the channelgroup
(type=int)

`name:` name of the channelgroup
(type=str)

`atype:` defines if the channelgroup is a templategroup (value==0) or a regular one (value==1)
(type=int)

`iconID:` id of the icon displayed for members or 0 if no icon is displayed
(type=int)

`saveDB:` set to 1 if memberships are saved to the server's database, otherwise set to 0
(type=int)

onChannelGroupListFinishedEvent(*self*, *serverConnectionHandlerID*)

This is called after each channelgroup yielded by onChannelGroupListEvent was triggered.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

onChannelGroupPermListEvent(*self*, *serverConnectionHandlerID*, *channelGroupID*, *permissionID*, *permissionValue*, *permissionNegated*, *permissionSkip*)

This is called for each granted permission assigned to a channelgroup requested with ts3lib.requestChannelGroupPermList.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelGroupID:` id of the channelgroup
(type=int)

`permissionID:` id of the permission
(type=int)

`permissionValue:` value of the permission
(type=int)

`permissionNegated:` negated flag of the permission
(type=int)

`permissionSkip:` skip flag of the permission
(type=int)

onChannelGroupPermListFinishedEvent(*self*, *serverConnectionHandlerID*, *channelGroupID*)

This is called after each permission yielded by onChannelGroupPermListEvent was triggered.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelGroupID:` id of the channelgroup
(type=int)

onChannelMoveEvent(*self*, *serverConnectionHandlerID*, *channelID*, *newChannelParentID*, *invokerID*, *invokerName*, *invokerUniqueIdentifier*)

This is called whenever a channel is moved to a new parent. If a channel is moved without changing the parent, onUpdateChannelEditedEvent is called instead.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	id of the moved channel *(type=int)*
`newChannelParentID:`	id of the new parent channel *(type=int)*
`invokerID:`	id of the moving client *(type=int)*
`invokerName:`	nick of the moving client *(type=str)*
`invokerUniqueIdentifier:`	uid of the moving client *(type=str)*

onChannelPasswordChangedEvent(*self*, *serverConnectionHandlerID*, *channelID*)

This is called whenever a channelpassword is changed.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	if of the channel *(type=int)*

onChannelPermListEvent(*self*, *serverConnectionHandlerID*, *channelID*, *permissionID*, *permissionValue*, *permissionNegated*, *permissionSkip*)

This is called for each granted permission of a channel requested by ts3lib.requestChannelPermList.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the id of the channel *(type=int)*
`permissionID:`	the id of the permission *(type=int)*
`permissionValue:`	the value of the permission *(type=int)*
`permissionNegated:`	negated flag of the permission *(type=int)*
`permissionSkip:`	skip flag of the permission *(type=int)*

onChannelPermListFinishedEvent(*self*, *serverConnectionHandlerID*, *channelID*)

This is called after each permission yielded by onChannelPermListEvent was triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the id of the channel *(type=int)*

onChannelSubscribeEvent(*self*, *serverConnectionHandlerID*, *channelID*)

This is called whenever a channel was subscribed.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the id of the channel *(type=int)*

onChannelSubscribeFinishedEvent(*self*, *serverConnectionHandlerID*)

This is called after the subscription on a server has finished (either after subscribing one channel, after all subscriptions of a channel family has been yielded by onChannelSubscribeEvent or after all subscriptions had been reset after connecting).

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

onChannelUnsubscribeEvent(*self*, *serverConnectionHandlerID*, *channelID*)

This is called whenever a channel was unsubscribed.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelID:` the id of the channel
(type=int)

onChannelUnsubscribeFinishedEvent(*self*, *serverConnectionHandlerID*)

This is called after the subscription on a server has finished (either after unsubscribing one channel or after all unsubscriptions of a channel family has been yielded by onChannelUnsubscribeEvent).

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

onClientBanFromServerEvent(*self*, *serverConnectionHandlerID*, *clientID*, *oldChannelID*, *newChannelID*, *visibility*, *kickerID*, *kickerName*, *kickerUniqueIdentifier*, *time*, *kickMessage*)

This is called after a client was banned from the server.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the id of the banned client *(type=int)*
`oldChannelID:`	the id of the last channel, the banned client was in *(type=int)*
`newChannelID:`	always set to 0 *(type=int)*
`visibility:`	always set to ts3defines.Visibility.LEAVE_VISIBILITY *(type=int)*
`kickerID:`	id of the banning client *(type=int)*
`kickerName:`	nick of the banning client *(type=str)*
`kickerUniqueIdentifier:`	uid of the banning client *(type=str)*
`time:`	duration of the ban in seconds *(type=int)*
`kickMessage:`	the kick and ban reason *(type=str)*

onClientChannelGroupChangedEvent(*self*, *serverConnectionHandlerID*, *channelGroupID*, *channelID*, *clientID*, *invokerClientID*, *invokerName*, *invokerUniqueIdentity*)

This is called whenever a client is added to a channelgroup in a specific channel.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelGroupID:`	the id of the channelgroup *(type=int)*
`channelID:`	the id of the channel *(type=int)*
`clientID:`	the id of the client *(type=int)*
`invokerClientID:`	the id of the client who added the channelgroup or 0 if the server did *(type=int)*
`invokerName:`	the nick of the client who added the channelgroup or "Server" if the server did *(type=str)*
`invokerUniqueIdentity:`	uid of the client who added the channelgroup or an empty string if the server did *(type=str)*

onClientChatClosedEvent(*self*, *serverConnectionHandlerID*, *clientID*, *clientUniqueIdentity*)

This is called after a client closed the chat to this client (but only after the other client has sent at least one message). This is either invoked by the sdk with ts3lib.clientChatClosed or the user has closed the conversation tab).

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the id of the other client *(type=int)*
`clientUniqueIdentity:`	the uid of the other client *(type=str)*

onClientChatComposingEvent(*self*, *serverConnectionHandlerID*, *clientID*, *clientUniqueIdentity*)

This is called whenever another client sends the chat composing command (either invoked by the sdk with ts3lib.clientChatComposing or when the user is really writing in the chat).

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	*(type=)*
`clientUniqueIdentity:`	*(type=)*

onClientDBIDfromUIDEvent(*self*, *serverConnectionHandlerID*, *uniqueClientIdentifier*, *clientDatabaseID*)

This is called whenever a database id was requested with ts3lib.requestClientDBIDfromUID.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`uniqueClientIdentifier:`	the uid of the requested client *(type=str)*
`clientDatabaseID:`	the resulting id in the database *(type=int)*

onClientDisplayNameChanged(*self*, *serverConnectionHandlerID*, *clientID*, *displayName*, *uniqueClientIdentifier*)

This is called whenever a client's displayname changed (nickname or friend/foe manager).

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the id of the client *(type=int)*
`displayName:`	the new displayname *(type=str)*
`uniqueClientIdentifier:`	the uid of the client *(type=str)*

onClientIDsEvent(*self*, *serverConnectionHandlerID*, *uniqueClientIdentifier*, *clientID*, *clientName*)

This is called for each client matching a specific uid requested by ts3lib.requestClientIDs.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`uniqueClientIdentifier:`	the uid of the client *(type=str)*
`clientID:`	the id of a client *(type=int)*
`clientName:`	the nick of the client *(type=str)*

onClientIDsFinishedEvent(*self*, *serverConnectionHandlerID*)

This is called after each client yielded by onClientIDsEvent was triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*

onClientKickFromChannelEvent(*self*, *serverConnectionHandlerID*, *clientID*, *oldChannelID*, *newChannelID*, *visibility*, *kickerID*, *kickerName*, *kickerUniqueIdentifier*, *kickMessage*)

This is called whenever a client is kicked from a channel.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the id of the kicked client *(type=int)*
`oldChannelID:`	the id of the channel the client was kicked from *(type=int)*
`newChannelID:`	the id of the channel the client was kicked to (the default channel) *(type=int)*
`visibility:`	defines the new state of the client in the view (see ts3defines.Visibility) *(type=int)*
`kickerID:`	the id of the kicking client *(type=int)*
`kickerName:`	the nick of the kicking client *(type=str)*
`kickerUniqueIdentifier:`	the uid of the kicking client *(type=str)*
`kickMessage:`	the kick reason *(type=str)*

onClientKickFromServerEvent(*self*, *serverConnectionHandlerID*, *clientID*, *oldChannelID*, *newChannelID*, *visibility*, *kickerID*, *kickerName*, *kickerUniqueIdentifier*, *kickMessage*)

This is called whenever a client is kicked from the server.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the id of the kicked client *(type=int)*
`oldChannelID:`	the id of the channel the client was in *(type=int)*
`newChannelID:`	always set to 0 *(type=int)*
`visibility:`	always set to ts3defines.Visibility.LEAVE_VISIBILITY *(type=int)*
`kickerID:`	the id of the kicking client *(type=int)*
`kickerName:`	nick of the kicking client *(type=str)*
`kickerUniqueIdentifier:`	uid of the kicking client *(type=str)*
`kickMessage:`	the kick reason *(type=str)*

onClientMoveEvent(*self*, *serverConnectionHandlerID*, *clientID*, *oldChannelID*, *newChannelID*, *visibility*, *moveMessage*)

This is called whenever a client enters a another channel (moving, joining or leaving the server).

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the id of the client
(type=int)

`oldChannelID:` the id of the former channel or 0 if the client joined the server
(type=int)

`newChannelID:` the id of the new channel or 0 if the client disconnected
(type=int)

`visibility:` defines the new state of the client in the view (see ts3defines.Visibility)
(type=int)

`moveMessage:` the disconnect message if the client left the server or an empty string
(type=str)

onClientMoveMovedEvent(*self*, *serverConnectionHandlerID*, *clientID*, *oldChannelID*, *newChannelID*, *visibility*, *moverID*, *moverName*, *moverUniqueIdentifier*, *moveMessage*)

This is called whenever a client is moved to another channel by another client.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the id of the moved client *(type=int)*
`oldChannelID:`	the id of the former channel *(type=int)*
`newChannelID:`	the id of the new channel *(type=int)*
`visibility:`	defines the new state of the client in the view (see ts3defines.Visibility) *(type=int)*
`moverID:`	the id of the moving client *(type=int)*
`moverName:`	nick of the moving client *(type=str)*
`moverUniqueIdentifier:`	uid of the moving client *(type=str)*
`moveMessage:`	always set to an empty string *(type=str)*

onClientMoveSubscriptionEvent(*self*, *serverConnectionHandlerID*, *clientID*, *oldChannelID*, *newChannelID*, *visibility*)

This is called whenever a new client enters the view when subscribing a channel.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the id of the client *(type=int)*
`oldChannelID:`	always set to 0 *(type=int)*
`newChannelID:`	the id of the subscribed channel *(type=int)*
`visibility:`	always set to ts3defines.Visibility.ENTER_VISIBILITY *(type=int)*

onClientMoveTimeoutEvent(*self*, *serverConnectionHandlerID*, *clientID*, *oldChannelID*, *newChannelID*, *visibility*, *timeoutMessage*)

This is called when a client timed out.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the id of the client *(type=int)*
`oldChannelID:`	the id of the channel the client was in *(type=int)*
`newChannelID:`	always set to 0 *(type=int)*
`visibility:`	always set to ts3defines.Visibility.LEAVE_VISIBILITY *(type=int)*
`timeoutMessage:`	the timeout message *(type=str)*

onClientNamefromDBIDEvent(*self*, *serverConnectionHandlerID*, *uniqueClientIdentifier*, *clientDatabaseID*, *clientNickName*)

This is called to return the last nickname of a client referenced by the database id after it was requested with ts3lib.requestClientNamefromDBID.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`uniqueClientIdentifier:`	the uid of the client *(type=str)*
`clientDatabaseID:`	the database id of the client *(type=int)*
`clientNickName:`	the last nickname of the client *(type=str)*

onClientNamefromUIDEvent(*self*, *serverConnectionHandlerID*, *uniqueClientIdentifier*, *clientDatabaseID*, *clientNickName*)

This is called to return the last nickname of a client referenced by the uid after it was requested with ts3lib.requestClientNamefromUID.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`uniqueClientIdentifier:`	the uid of the client *(type=str)*
`clientDatabaseID:`	the database id of the client *(type=int)*
`clientNickName:`	the last nickname of the client *(type=str)*

onClientNeededPermissionsEvent(*self*, *serverConnectionHandlerID*, *permissionID*, *permissionValue*)

This is called whenever a permission the TS3 client needed changes.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`permissionID:`	the id of the permission *(type=int)*
`permissionValue:`	the value of the permission *(type=int)*

onClientNeededPermissionsFinishedEvent(*self*, *serverConnectionHandlerID*)

This is called after each permission yielded by onClientNeededPermissionsEvent was triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*

onClientPermListEvent(*self*, *serverConnectionHandlerID*, *clientDatabaseID*, *permissionID*, *permissionValue*, *permissionNegated*, *permissionSkip*)

This is called for each granted permission to a specific client requested with ts3lib.requestClientPermList.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientDatabaseID:`	the database id of the client *(type=int)*
`permissionID:`	the id of the permission *(type=int)*
`permissionValue:`	the value of the permission *(type=int)*
`permissionNegated:`	negated flag of the permission *(type=int)*
`permissionSkip:`	skip flag of the permission *(type=int)*

onClientPermListFinishedEvent(*self*, *serverConnectionHandlerID*, *clientDatabaseID*)

This is called after each permission yielded by onClientPermListEvent was triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientDatabaseID:`	the database id of the client *(type=int)*

onClientSelfVariableUpdateEvent(*self*, *serverConnectionHandlerID*, *flag*, *oldValue*, *newValue*)

This is called whenever a variable of the own client is changed.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`flag:`	the changed variable (see ts3defines.ClientProperties and ts3defines.ClientPropertiesRare) *(type=int)*
`oldValue:`	the former value *(type=str)*
`newValue:`	the new value *(type=str)*

onClientServerQueryLoginPasswordEvent(*self*, *serverConnectionHandlerID*, *loginPassword*)

This is called when a new query login was requested.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`loginPassword:`	the new password *(type=str)*

onComplainListEvent(*self*, *serverConnectionHandlerID*, *targetClientDatabaseID*, *targetClientNickName*, *fromClientDatabaseID*, *fromClientNickName*, *complainReason*, *timestamp*)

This is called for each entry in the complaintslist.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`targetClientDatabaseID:` the database id of the complained client
(type=int)

`targetClientNickName:` the last nickname of the complained client
(type=str)

`fromClientDatabaseID:` the database id of the complaining client
(type=int)

`fromClientNickName:` the last nickname of the complaining client
(type=str)

`complainReason:` the reason
(type=str)

`timestamp:` the time of the complain as unix timestamp
(type=int)

onConnectionInfoEvent(*self*, *serverConnectionHandlerID*, *clientID*)

This is called when the connection info of a client has been updated requested with ts3lib.requestConnectionInfo.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientID:` the id of the client
(type=int)

onConnectStatusChangeEvent(*self*, *serverConnectionHandlerID*, *newStatus*, *errorNumber*)

This is called whenever the status of a serverconnection changed.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`newStatus:`	the new status (see ts3defines.ConnectStatus) *(type=int)*
`errorNumber:`	the error (see ts3defines.ERROR_*) *(type=int)*

onDelChannelEvent(*self*, *serverConnectionHandlerID*, *channelID*, *invokerID*, *invokerName*, *invokerUniqueIdentifier*)

This is called whenever a channel was deleted.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the id of the channel *(type=int)*
`invokerID:`	the id of the client who deleted the channel *(type=int)*
`invokerName:`	the nick of the deleting client *(type=str)*
`invokerUniqueIdentifier:`	the uid of the deleting client *(type=str)*

onFileInfoEvent(*self*, *serverConnectionHandlerID*, *channelID*, *name*, *size*, *datetime*)

This is called with the fileinfo of a remote file requested with ts3lib.requestFileInfo.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the id of the channel the file is in *(type=int)*
`name:`	the full path of the file *(type=str)*
`size:`	the filesize in bytes *(type=int)*
`datetime:`	time the file was last changed as unix timestamp *(type=int)*

onFileListEvent(*self*, *serverConnectionHandlerID*, *channelID*, *path*, *name*, *size*, *datetime*, *atype*, *incompletesize*, *returnCode*)

This is called for each file and directory in path requested with ts3lib.requestFileList.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the id of the channel *(type=int)*
`path:`	the path *(type=str)*
`name:`	the filename *(type=str)*
`size:`	the filesize in bytes *(type=int)*
`datetime:`	time the file or directory was last changed as unix timestamp *(type=)*
`atype:`	set to 1 if it's a directory, otherwise set to 0 *(type=int)*
`incompletesize:`	the complete filesize in bytes or 0 if the file is already complete *(type=int)*
`returnCode:`	the returncode passed to the request or an empty string *(type=str)*

onFileListFinishedEvent(*self*, *serverConnectionHandlerID*, *channelID*, *path*)

This is called after each file and directory yielded by onFileListEvent was triggered.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelID:` *(type=int)*

`path:` *(type=)*

onHotkeyEvent(*self*, *keyword*)

This is called when a plugin's hotkey is triggered.

Parameters

`keyword:` the local keyword set in cls.hotkeys
(type=str)

onHotkeyRecordedEvent(*self*, *keyword*, *key*)

This is called when a hotkey was recorded requested by ts3lib.requestHotkeyInputDialog.

Parameters

`keyword:` the keyword
(type=str)

`key:` the hotkey to trigger the keyword
(type=str)

onIncomingClientQueryEvent(*self*, *serverConnectionHandlerID*, *commandText*)

This callback was designed for the clientquery plugin. It combines many callbacks and is called with a representing string.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`commandText:` the text of the client query
(type=str)

onMenuItemEvent(*self*, *serverConnectionHandlerID*, *atype*, *menuItemID*, *selectedItemID*)

This is called when a plugin's menuitem defined in cls.menuItems is clicked.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`atype:`	type of the menuitem (see ts3defines.PluginMenuType) *(type=int)*
`menuItemID:`	the local id of the menuitem defines in cls.menuItems *(type=int)*
`selectedItemID:`	set to the id of the selected channel if atype is ts3defines.PluginMenuType.PLUGIN_MENU_TYPE_CHANNE set to the id of the selected client if atype is ts3defines.PluginMenuType.PLUGIN_MENU_TYPE_CLIENT, otherwise always set to 0 *(type=int)*

onMessageGetEvent(*self*, *serverConnectionHandlerID*, *messageID*, *fromClientUniqueIdentity*, *subject*, *message*, *timestamp*)

This is called with the information about an offline message requested with ts3lib.requestMessageGet.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`messageID:`	the id of the message *(type=int)*
`fromClientUniqueIdentity:`	the uid of the message's sender *(type=str)*
`subject:`	the subject of the message *(type=str)*
`message:`	the content of the message *(type=str)*
`timestamp:`	time the message was sent as unix timestamp *(type=int)*

onMessageListEvent(*self*, *serverConnectionHandlerID*, *messageID*, *fromClientUniqueIdentity*, *subject*, *timestamp*, *flagRead*)

This is called for each offline message available on the server requested with ts3lib.requestMessageList.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`messageID:`	the id of the message *(type=int)*
`fromClientUniqueIdentity:`	the uid of the message's sender *(type=str)*
`subject:`	the subject of the message *(type=str)*
`timestamp:`	time the message was sent as unix timestamp *(type=int)*
`flagRead:`	defines the read status of the message *(type=int)*

onNewChannelCreatedEvent(*self*, *serverConnectionHandlerID*, *channelID*, *channelParentID*, *invokerID*, *invokerName*, *invokerUniqueIdentifier*)

This is called whenever a new channel was created.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the id of the new channel *(type=int)*
`channelParentID:`	the id of the parent channel *(type=int)*
`invokerID:`	the id of the creating client *(type=int)*
`invokerName:`	nick of the creating client *(type=str)*
`invokerUniqueIdentifier:`	the uid of the creating client *(type=str)*

onNewChannelEvent(*self*, *serverConnectionHandlerID*, *channelID*, *channelParentID*)

This is called whenever a new channel enters the view (at connect).

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`channelID:`	the id of the channel *(type=int)*
`channelParentID:`	the id of the parent channel *(type=int)*

onPermissionListEvent(*self*, *serverConnectionHandlerID*, *permissionID*, *permissionName*, *permissionDescription*)

This is called for each permission on the server requested with ts3lib.requestPermissionList.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`permissionID:`	id of the permission *(type=int)*
`permissionName:`	name of the permission *(type=str)*
`permissionDescription:`	description of the permission *(type=str)*

onPermissionListFinishedEvent(*self*, *serverConnectionHandlerID*)

This is called after each permission yielded by onPermissionListEvent was triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*

onPermissionListGroupEndIDEvent(*self*, *serverConnectionHandlerID*, *groupEndID*)

This is called for each last permission in the groups of permissions after requesting the permissionlist with ts3lib.requestPermissionList.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`groupEndID:`	id of the last permission in group *(type=int)*

onPermissionOverviewEvent(*self*, *serverConnectionHandlerID*, *clientDatabaseID*, *channelID*, *overviewType*, *overviewID1*, *overviewID2*, *permissionID*, *permissionValue*, *permissionNegated*, *permissionSkip*)

This is called for each permission of a pair of client and channel requested with ts3lib.requestPermissionOverview.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientDatabaseID:`	the database id of the client *(type=int)*
`channelID:`	the id of the channel *(type=int)*
`overviewType:`	defines the type of entry in the overview (0 for servergroup, 1 for client permissions, 2 for needed channel permissions, 3 for channelgroup) *(type=int)*
`overviewID1:`	depending on the overviewType, set to the id of the servergroup, to the client's database id or the id of the channel *(type=int)*
`overviewID2:`	only used with overviewType=3, then set to the id of the channelgroup; otherwise set to 0 *(type=int)*
`permissionID:`	the id of the permission *(type=int)*
`permissionValue:`	the value of the permission *(type=int)*
`permissionNegated:`	negated flag of the permission *(type=int)*
`permissionSkip:`	skip flag of the permission *(type=int)*

onPermissionOverviewFinishedEvent(*self*, *serverConnectionHandlerID*)

This is called after each permission yielded by onPermissionOverviewEvent was triggered.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

onPlaybackShutdownCompleteEvent(*self*, *serverConnectionHandlerID*)

This is called when a playback device can be shutdown with ts3lib.closePlaybackDevice after the process was initiated with ts3lib.initiateGracefulPlaybackShutdown.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

onPluginCommandEvent(*self*, *serverConnectionHandlerID*, *pluginName*, *pluginCommand*)

This is called whenever pyTSon receives a plugincommand from another client. All pyTSon plugins will receive this callback. pyTSon recommends to prefix plugincommands with the pluginname.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`pluginName:` the name of the sending plugin
(type=str)

`pluginCommand:` the command
(type=str)

onServerConnectionInfoEvent(*self*, *serverConnectionHandlerID*)

This is called whenever the server's connectioninfo was updated.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

onServerEditedEvent(*self*, *serverConnectionHandlerID*, *editerID*, *editerName*, *editerUniqueIdentifier*)

This is called whenever the server was edited by a client.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`editerID:`	the id of the client *(type=int)*
`editerName:`	nick of the client *(type=int)*
`editerUniqueIdentifier:`	uid of the client *(type=str)*

onServerGroupByClientIDEvent(*self*, *serverConnectionHandlerID*, *name*, *serverGroupList*, *clientDatabaseID*)

This is called for each servergroup of a client requested with ts3lib.requestServerGroupsByClientID.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`name:`	name of the servergroup *(type=str)*
`serverGroupList:`	id of the servergroup *(type=int)*
`clientDatabaseID:`	the database id of the client *(type=int)*

onServerGroupClientAddedEvent(*self*, *serverConnectionHandlerID*, *clientID*, *clientName*, *clientUniqueIdentity*, *serverGroupID*, *invokerClientID*, *invokerName*, *invokerUniqueIdentity*)

This is called whenever a client is added to a servergroup.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the id of the added client *(type=int)*
`clientName:`	nick of the added client *(type=str)*
`clientUniqueIdentity:`	uid of the added client *(type=str)*
`serverGroupID:`	the id of the servergroup *(type=int)*
`invokerClientID:`	the id of the adding client *(type=int)*
`invokerName:`	nick of the adding client *(type=str)*
`invokerUniqueIdentity:`	uid of the adding client *(type=str)*

onServerGroupClientDeletedEvent(*self*, *serverConnectionHandlerID*, *clientID*, *clientName*, *clientUniqueIdentity*, *serverGroupID*, *invokerClientID*, *invokerName*, *invokerUniqueIdentity*)

This is called whenever a client was removed from a servergroup.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the id of the removed client *(type=int)*
`clientName:`	nick of the removed client *(type=str)*
`clientUniqueIdentity:`	uid of the removed client *(type=str)*
`serverGroupID:`	id the servergroup *(type=int)*
`invokerClientID:`	the id of the removing client *(type=int)*
`invokerName:`	nick of the removing client *(type=str)*
`invokerUniqueIdentity:`	uid of the removing client *(type=str)*

onServerGroupClientListEvent(*self*, *serverConnectionHandlerID*, *serverGroupID*, *clientDatabaseID*, *clientNameIdentifier*, *clientUniqueID*)

This is called for each member of a servergroup requested with ts3lib.requestServerGroupClientList.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`serverGroupID:` the id of the servergroup
(type=int)

`clientDatabaseID:` the database id of the member
(type=int)

`clientNameIdentifier:` the last nick of the member or an empty string if withNames was set to False in the request
(type=str)

`clientUniqueID:` the uid of the member or an empty string if withNames was set to False in the request
(type=str)

onServerGroupListEvent(*self*, *serverConnectionHandlerID*, *serverGroupID*, *name*, *atype*, *iconID*, *saveDB*)

This is called for each servergroup on the server requested with ts3lib.requestServerGroupList.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`serverGroupID:` the id of the servergroup
(type=int)

`name:` name of the servergroup
(type=str)

`atype:` type of the servergroup (0=template, 1=regular, 2=serverquery)
(type=int)

`iconID:` icon id of the servergroup or 0 if no icon in this group
(type=int)

`saveDB:` set to 1 if memberships are saved to the database, set to 0 otherwise
(type=int)

onServerGroupListFinishedEvent(*self*, *serverConnectionHandlerID*)

This is called after each servergroup yielded by onServerGroupListEvent was triggered.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

onServerGroupPermListEvent(*self*, *serverConnectionHandlerID*, *serverGroupID*, *permissionID*, *permissionValue*, *permissionNegated*, *permissionSkip*)

This is called for each granted permission of a servergroup requested with ts3lib.requestServerGroupPermList.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`serverGroupID:`	the id of the servergroup *(type=int)*
`permissionID:`	the id of the permission *(type=int)*
`permissionValue:`	value of the permission *(type=int)*
`permissionNegated:`	negated flag *(type=int)*
`permissionSkip:`	skip flag *(type=int)*

onServerGroupPermListFinishedEvent(*self*, *serverConnectionHandlerID*, *serverGroupID*)

This is called after each permission yielded by onServerGroupPermListEvent was triggered.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`serverGroupID:`	id of the servergroup *(type=int)*

onServerLogEvent(*self*, *serverConnectionHandlerID*, *logMsg*)

This is called for each line of the serverlog requested by the TS3 Client.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`logMsg:` the message
(type=str)

onServerLogFinishedEvent(*self*, *serverConnectionHandlerID*, *lastPos*, *fileSize*)

This is called after the requested number of loglines were yielded by onServerLogEvent.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`lastPos:` *(type=)*

`fileSize:` *(type=)*

onServerStopEvent(*self*, *serverConnectionHandlerID*, *shutdownMessage*)

This is called when the server was stopped.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`shutdownMessage:` if given, the shutdownmessage
(type=str)

onServerTemporaryPasswordListEvent(*self*, *serverConnectionHandlerID*, *clientNickname*, *uniqueClientIdentifier*, *description*, *password*, *timestampStart*, *timestampEnd*, *targetChannelID*, *targetChannelPW*)

This is called for each temporary password on the server requested with ts3lib.requestServerTemporaryPasswordList.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`clientNickname:` nick of the creator
(type=str)

`uniqueClientIdentifier:` uid of the creator
(type=str)

`description:` description of the password
(type=str)

`password:` the password
(type=str)

`timestampStart:` time the password was created as unix timestamp
(type=int)

`timestampEnd:` time the password expires as unix timestamp
(type=int)

`targetChannelID:` the id of the channel clients join in
(type=int)

`targetChannelPW:` password to the targetChannel
(type=str)

onServerUpdatedEvent(*self*, *serverConnectionHandlerID*)

This is called whenever the server variables were updated.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

onSoundDeviceListChangedEvent(*self*, *modeID*, *playOrCap*)

This is called when the list of sounddevices changed.

Parameters

`modeID:` defines the playback/capture mode

(type=int)

`playOrCap:` defines whether the playback- or capturelist changed

(type=int)

onTalkStatusChangeEvent(*self*, *serverConnectionHandlerID*, *status*, *isReceivedWhisper*, *clientID*)

This is called whenever a client starts or stops talking.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection

(type=int)

`status:` defines whether the client starts or stops talking (see ts3defines.TalkStatus)

(type=int)

`isReceivedWhisper:` set to 1 if the client whispered, set to 0 otherwise

(type=int)

`clientID:` the id of the client

(type=int)

onUpdateChannelEditedEvent(*self*, *serverConnectionHandlerID*, *channelID*, *invokerID*, *invokerName*, *invokerUniqueIdentifier*)

This is called whenever a channel was edited by a client.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelID:` the id of the channel
(type=int)

`invokerID:` the id of the client
(type=int)

`invokerName:` nick of the client
(type=str)

`invokerUniqueIdentifier:` uid of the client
(type=str)

onUpdateChannelEvent(*self*, *serverConnectionHandlerID*, *channelID*)

This is called whenever the channel variables of a specific channel are updated.

Parameters

`serverConnectionHandlerID:` the ID of the serverconnection
(type=int)

`channelID:` the id of the channel
(type=int)

onUpdateClientEvent(*self*, *serverConnectionHandlerID*, *clientID*, *invokerID*, *invokerName*, *invokerUniqueIdentifier*)

This is called whenever the client variables of a specific client are updated.

Parameters

`serverConnectionHandlerID:`	the ID of the serverconnection *(type=int)*
`clientID:`	the id of the client *(type=int)*
`invokerID:`	id of the client invoking the change or 0 if it was a selfupdate *(type=int)*
`invokerName:`	nick of the invoking client *(type=str)*
`invokerUniqueIdentifier:`	uid of the invoking client *(type=str)*

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

14.2.2 Properties

Name	Description
Inherited from object __class__	

14.2.3 Class Variables

Name	Description
requestAutoload	If set to True, the plugin is automatically loaded on startup. This check is only done once per new plugin, after that users can enable/disable the plugin. **Value:** `False`
name	The name of the plugin. Use meaningful names. It has to be unique in the list of plugins. **Value:** `"__ts3plugin__"`

continued on next page

Name	Description
version	Version string of the plugin. pyTSon will use this string to determine, if a new version is available in an online repository. **Value:** `"1.0"`
apiVersion	apiVersion the plugin was developed for. **Value:** `21`
author	Let the world know who made the plugin. **Value:** `"Thomas \"PLuS\" Pathmann"`
description	Explain, what the plugin does. **Value:** `"This is the baseclass for all ts3 python plugins"`
offersConfigure	Set this to True, if the plugin offers a configuration ui. In this case the method configure is called. **Value:** `False`
commandKeyword	Set this to a keyword (non-empty) your plugin can be called with. Users may type /py <thecommand> [moreargs]. The method processCommand will be called with any additional args. **Value:** `"py"`
infoTitle	If set to a string, this title is shown in the info frame of the client on top of the infoData. If set to None, nothing is shown and infoData won't be called. **Value:** `"pyTSon"`
menuItems	List of tuple(int, int, str, str) containing the menuitems. The tuple has to contain the type (see ts3defines.PluginMenuType), an int identifier (unique in this list), the title and the name of the icon. The icon has to be a path relative to pytson.getPluginPath(). Pass an empty string to omit using an icon. The method onMenuItemEvent with menuItemID=identifier is called. **Value:** `[(ts3defines.PluginMenuType.PLUGIN_MENU_TYPE_CLIENT, 0, "...`

continued on next page

Name	Description
hotkeys	List of tuple(str, str) containing the hotkeys. The tuple has to contain a string identifier (unique in this list) and a description shown in the TS3 Client's hotkey dialog. The method onHotkeyEvent with keyword=identifier is called. **Value:** `[("keyword", "description")]`

15 Package ts3widgets

15.1 Modules

- **filetransfer** *(Section 16, p. 213)*
- **serverview** *(Section 17, p. 233)*

16 Module ts3widgets.filetransfer

16.1 Functions

splitpath(*path*)

Splits a TS3 filepath into its sections.

Parameters

`path`: the path to split

(type=str)

Return Value

the list of sections

(type=list[str])

joinpath(**args*)

Joins multiple sections into a TS3 filepath.

Parameters

`args`: sections to join

(type=tuple(str))

Return Value

the resulting path

(type=str)

bytesToStr(*size*)

Creates a human readable string of a number of bytes.

Parameters

`size`: number of bytes

(type=int)

Return Value

the converted size and most fitting unit

(type=str)

16.2 Class File

object
└ **ts3widgets.filetransfer.File**

Container class to hold all information on a remote TS3 file.

16.2.1 Methods

__init__(*self*, *path*, *name*, *size*, *date*, *atype*, *incompletesize*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

isDirectory(*self*)

icon(*self*)

Returns the most fitting icon for the file

Return Value

the icon

(type=QIcon)

fullpath(*self*)

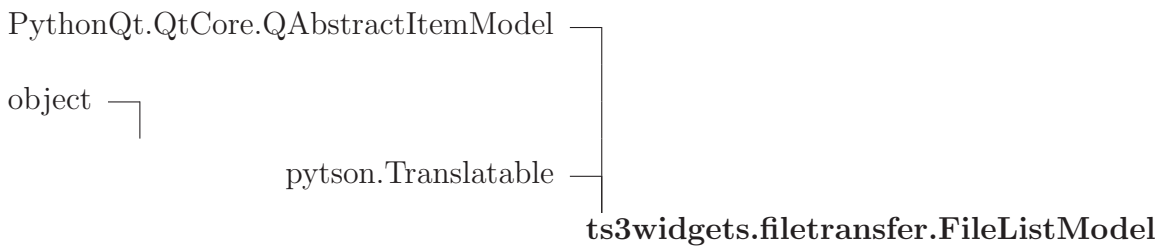
Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.2.2 Properties

Name	Description
Inherited from object	
__class__	

16.3 Class FileListModel



Itemmodel to abstract the files contained on a TS3 filepath.

16.3.1 Methods

__init__(*self*, *schid*, *cid*, *password*, *parent*=`None`, *readonly*=`False`)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__del__(*self*)

path(*self*, *val*)

currentFiles(*self*)

onFileListEvent(*self*, *schid*, *channelID*, *path*, *name*, *size*, *date*, *atype*, *incompletesize*, *returnCode*)

onFileListFinishedEvent(*self*, *schid*, *channelID*, *path*)

onServerErrorEvent(*self*, *schid*, *errorMessage*, *error*, *returnCode*, *extraMessage*)

onServerPermissionErrorEvent(*self*, *schid*, *errorMessage*, *error*, *returnCode*, *failedPermissionID*)

headerData(*self*, *section*, *orientation*, *role*=`Qt.DisplayRole`)

flags(*self*, *idx*)

index(*self*, *row*, *column*, *parent*=`QModelIndex()`)

parent(*self*, *idx*)

rowCount(*self*, *parent*=`QModelIndex()`)

columnCount(*self*, *parent*=`QModelIndex()`)

data(*self*, *idx*, *role*=`Qt.DisplayRole`)

setData(*self*, *idx*, *value*, *role*=`Qt.EditRole`)

fileByIndex(*self*, *idx*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.3.2 Properties

Name	Description
Inherited from object	
__class__	

16.4 Class SmartStatusBar

PythonQt.QtGui.QStatusBar
└ **ts3widgets.filetransfer.SmartStatusBar**

StatusBar which automatically hides itsself, when the message is cleared.

16.4.1 Methods

__**init**__(*self*, *parent*=`None`)

showMessage(*self*, *message*, *timeout*=`0`)

Displays a message for a specified duration.

Parameters

`message`: the message to display
(type=str)

`timeout`: duration in ms; optional; if set to 0, SmartStatusBar.defaultTimeout is used
(type=int)

16.4.2 Class Variables

Name	Description
defaultTimeout	**Value:** `5000`

16.5 Class FileCollector

object
pytson.Translatable
ts3widgets.filetransfer.FileCollector

Collects all files recursively from TS3 filetransfer directories with their corresponding download path. Emits a signal collectionFinished with a list of tuples(str, list[File]) containing the download dir and a list of files. The signal collectionError(str, int) is emitted on error with the errorstring and the errorcode.

16.5.1 Methods

__init__(*self*, *schid*, *cid*, *password*, *rootdir*)

Instantiates a new object.

Parameters

`schid:` the id of the serverconnection handler
(type=int)

`cid:` the id of the channel
(type=int)

`password:` the password of the channel
(type=str)

`rootdir:` the root download directory
(type=str)

Overrides: object.__init__

__del__(*self*)

addFiles(*self*, *files*)

Manually adds a list of files to the collection (emitted with the rootdir)

Parameters

`files`: list of files to emit

(type=list(File))

collect(*self*, *dirs*)

Starts collecting files from a list of directories

Parameters

`dirs`: list of directories

(type=list(File))

onServerErrorEvent(*self*, *schid*, *errorMessage*, *error*, *returnCode*, *extraMessage*)

onFileListEvent(*self*, *schid*, *channelID*, *path*, *name*, *size*, *datetime*, *atype*, *incompletesize*, *returnCode*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.5.2 Properties

Name	Description
Inherited from object	
__class__	

16.6 Class FileBrowser

Dialog to display files contained on a TS3 filepath.

16.6.1 Methods

__init__(*self*, *schid*, *cid*, *password*=`''`, *path*=`'/'`, *parent*=`None`, *staticpath*=`False`, *readonly*=`False`, *downloaddir*=`None`, *iconpack*=`None`)

Instantiates a new object.

Parameters

`schid`: the id of the serverconnection handler

(type=int)

`cid`: the id of the channel

(type=int)

`password`: password to the channel, defaults to an empty string

(type=str)

`path`: path to display, defaults to the root path

(type=str)

`parent`: parent of the dialog; optional keyword arg; defaults to None

(type=QWidget)

`staticpath`: if set to True, the initial path can't be changed by the user; optional keyword arg; defaults to False

(type=bool)

`readonly`: if set to True, the user can't download, upload or delete files, or create new directories; optional keyword arg; defaults to False

(type=bool)

`downloaddir`: directory to download files to; optional keyword arg; defaults to None; if set to None, the TS3 client's download directory is used

(type=str)

`iconpack`: iconpack to load icons from; optional keyword arg; defaults to None; if set to None, the current iconpack is used

(type=ts3client.IconPack)

Overrides: object.__init__

__del__(*self*)

onPathChanged(*self*, *newpath*)

on_pathEdit_returnPressed(*self*)

on_iconButton_toggled(*self*, *act*)

on_detailedButton_toggled(*self*, *act*)

on_filterButton_clicked(*self*)

on_clearButton_clicked(*self*)

on_filterEdit_textChanged(*self*, *newtext*)

on_upButton_clicked(*self*)

on_homeButton_clicked(*self*)

refresh(*self*)

on_downloaddirButton_clicked(*self*)

showError(*self*, *prefix*, *errcode*, *msg*=`None`)

uploadFiles(*self*)

onServerErrorEvent(*self*, *schid*, *errorMessage*, *error*, *returnCode*, *extraMessage*)

selectedFiles(*self*)

currentItem(*self*, *source*=`True`)

downloadFiles(*self*, *files*=`None`)

createFolder(*self*)

deleteFiles(*self*, *files*=`None`)

on_table_customContextMenuRequested(*self*, *pos*)

on_list_customContextMenuRequested(*self*, *pos*)

viewDoubleClicked(*self*, *idx*)

on_openAction_triggered(*self*)

on_renameAction_triggered(*self*)

on_copyAction_triggered(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.6.2 Properties

Name	Description
Inherited from object	
__class__	

16.7 Class FileCollisionAction

object
└ **ts3widgets.filetransfer.FileCollisionAction**

16.7.1 Methods

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __init__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.7.2 Properties

Name	Description
Inherited from object	

continued on next page

Name	Description
__class__	

16.7.3 Class Variables

Name	Description
overwrite	**Value:** `1`
resume	**Value:** `2`
skip	**Value:** `4`
toall	**Value:** `8`

16.8 Class FileCollisionDialog

Dialog to inform about a filecollision and requests input how to handle it.

16.8.1 Methods

getAction(*cls*, *localfile*, *remotefile*, *isdownload*, *multi*, *parent*=`None`)

Convenience function to execute (blocks) the dialog.

Parameters

`localfile`: the path to the local file
(type=str)

`remotefile`: the remote file
(type=File)

`isdownload`: set to True if remotefile should be downloaded
(type=bool)

`multi`: set to True, if there are multiple files which could collide
(type=bool)

`parent`: parent widget of the dialog; optional; defaults to None
(type=QWidget)

__init__(*self*, *localfile*, *remotefile*, *isdownload*, *multi*, *parent*=`None`)

Instantiates a new dialog.

Parameters

`localfile`: the path to the local file
(type=str)

`remotefile`: the remote file
(type=File)

`isdownload`: set to True if remotefile should be downloaded
(type=bool)

`multi`: set to True, if there are multiple files which could collide
(type=bool)

`parent`: parent widget of the dialog; optional; defaults to None
(type=QWidget)

Overrides: object.__init__

on_overwriteButton_clicked(*self*)

on_resumeButton_clicked(*self*)

on_skipButton_clicked(*self*)

on_skipallButton_clicked(*self*)

on_cancelButton_clicked(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.8.2 Properties

Name	Description
Inherited from object	
__class__	

16.9 Class FileTransfer

object

pytson.Translatable

ts3widgets.filetransfer.FileTransfer

Abstract container class to hold information on a filetransfer

16.9.1 Methods

__init__(*self*, *err*, *retcode*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

updateSize(*self*, *val*)

updateError(*self*, *err*, *msg*=`None`)

progress(*self*)

hasError(*self*)

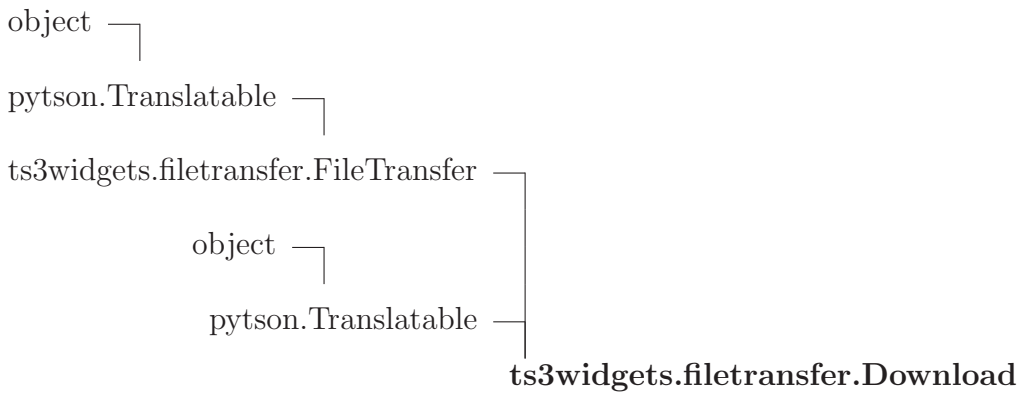
Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.9.2 Properties

Name	Description
Inherited from object	
__class__	

16.10 Class Download



Container class to hold information on a download

16.10.1 Methods

__init__(*self*, *err*, *retcode*, *thefile*, *todir*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

progress(*self*)

Overrides: ts3widgets.filetransfer.FileTransfer.progress

description(*self*)

localpath(*self*)

Inherited from ts3widgets.filetransfer.FileTransfer(Section 16.9)

hasError(), updateError(), updateSize()

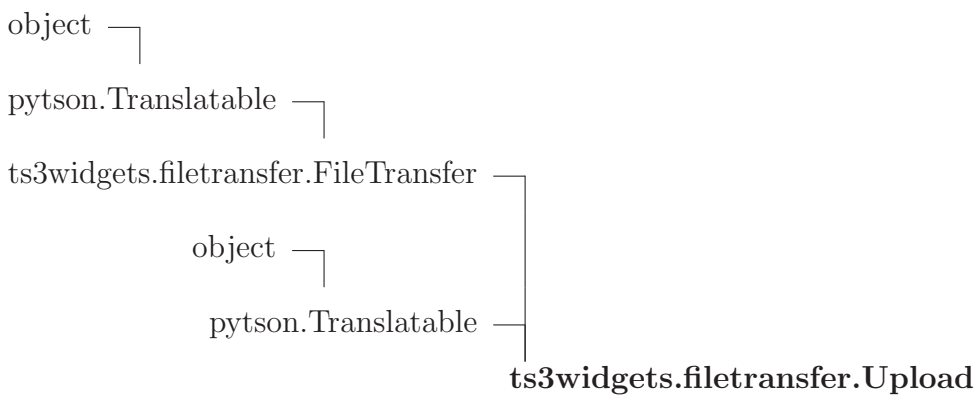
Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.10.2 Properties

Name	Description
Inherited from object	
__class__	

16.11 Class Upload



Container class to hold information on an upload

16.11.1 Methods

__init__(*self*, *err*, *retcode*, *localfile*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

progress(*self*)

Overrides: ts3widgets.filetransfer.FileTransfer.progress

description(*self*)

Inherited from ts3widgets.filetransfer.FileTransfer(Section 16.9)

hasError(), updateError(), updateSize()

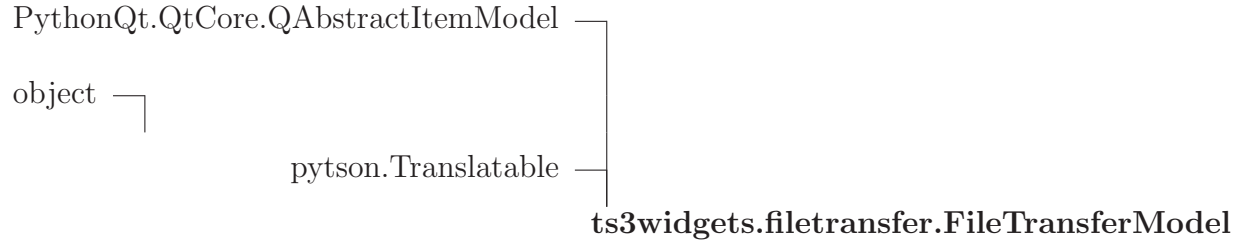
Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.11.2 Properties

Name	Description
Inherited from object	
__class__	

16.12 Class FileTransferModel



Itemmodel to abstract multiple filetransfers.

16.12.1 Methods

__init__(*self*, *schid*, *cid*, *password*, *parent*=`None`)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

__del__(*self*)

timerEvent(*self*, *event*)

addDownload(*self*, *thefile*, *downloaddir*, *overwrite*, *resume*)

Requests a download from the server and monitors its progress

Parameters

`thefile:` remote file to download
(type=File)

`downloaddir:` path to the download directory
(type=str)

`overwrite:` set to True to overwrite an existing file
(type=bool)

`resume:` set to True to resume a previous download
(type=bool)

Return Value

the filetransfer id
(type=int)

addUpload(*self*, *path*, *localfile*, *overwrite*, *resume*)

Requests an upload to the server.

Parameters

`path:` path to upload the file to
(type=str)

`localfile:` path to the file to upload
(type=str)

`overwrite:` set to True to overwrite an existing file
(type=bool)

`resume:` set to True to resume a previous upload
(type=bool)

Return Value

the filetransfer id
(type=int)

cleanup(*self*)

Cleanup finished and broken downloads

onFileTransferStatusEvent(*self*, *transferID*, *status*, *statusMessage*, *remotefileSize*, *schid*)

onServerErrorEvent(*self*, *schid*, *errorMessage*, *error*, *returnCode*, *extraMessage*)

onServerPermissionErrorEvent(*self*, **args*)

headerData(*self*, *section*, *orientation*, *role*=`Qt.DisplayRole`)

index(*self*, *row*, *column*, *parent*=`QModelIndex()`)

parent(*self*, *idx*)

rowCount(*self*, *parent*=`QModelIndex()`)

columnCount(*self*, *parent*=`QModelIndex()`)

data(*self*, *idx*, *role*=`Qt.DisplayRole`)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.12.2 Properties

Name	Description
Inherited from object	
__class__	

16.13 Class FileTransferDelegate

PythonQt.QtGui.QStyledItemDelegate
└ **ts3widgets.filetransfer.FileTransferDelegate**

Delegate which displays a progress bar in the second column of an itemview

16.13.1 Methods

paint(*self*, *painter*, *option*, *idx*)

16.14 Class FileTransferDialog

Dialog to display filetransfers from/to a ts3 channel.

16.14.1 Methods

__init__(*self*, *schid*, *cid*, *password*, *parent*=`None`)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

on_closeButton_clicked(*self*)

on_cleanupButton_clicked(*self*)

addUpload(*self*, *path*, *localfile*, *overwrite*, *resume*)

Adds an upload.

Parameters

`path:` path to upload the file to
(type=str)

`localfile:` path to the file to upload
(type=str)

`overwrite:` set to True to overwrite an existing file
(type=bool)

`resume:` set to True to resume a previous upload
(type=bool)

Return Value

the filetransfer id
(type=int)

addDownload(*self*, *thefile*, *downloaddir*, *overwrite*, *resume*)

Adds a download.

Parameters

`thefile:` remote file to download
(type=File)

`downloaddir:` path to the download directory
(type=str)

`overwrite:` set to True to overwrite an existing file
(type=bool)

`resume:` set to True to resume a previous download
(type=bool)

Return Value

the filetransfer id
(type=int)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.14.2 Properties

Name	Description
Inherited from object	
__class__	

17 Module ts3widgets.serverview

17.1 Class ServerViewRoles

Additional roles used in ServerviewModel to deliver icons and spacer properties.

17.1.1 Class Variables

Name	Description
itemtype	**Value:** `Qt.UserRole`
statusicons	**Value:** `Qt.UserRole+ 1`
isspacer	**Value:** `Qt.UserRole+ 2`
spacertype	**Value:** `Qt.UserRole+ 3`
spaceralignment	**Value:** `Qt.UserRole+ 4`
spacercustomtext	**Value:** `Qt.UserRole+ 5`

17.2 Class Channel

object
└ **ts3widgets.serverview.Channel**

Object wrapper for a channel on a TS3 server.

17.2.1 Methods

__init__(*self*, *schid*, *cid*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

append(*self*, *obj*, *sort*=`True`)

rowOf(*self*, *obj*=`None`, *pretend*=`False`)

remove(*self*, *obj*)

update(*self*)

name(*self*)

sortOrder(*self*)

isPermanent(*self*)

isSpacer(*self*)

spacerAlignment(*self*)

spacerType(*self*)

spacerCustomtext(*self*)

isPasswordProtected(*self*)

isSubscribed(*self*)

neededTalkPower(*self*)

isDefault(*self*)

iconID(*self*)

maxClients(*self*)

codec(*self*)

isFull(*self*)

iconVariable(*self*)

count(*self*)

child(*self*, *row*)

sort(*self*)

__iter__(*self*)

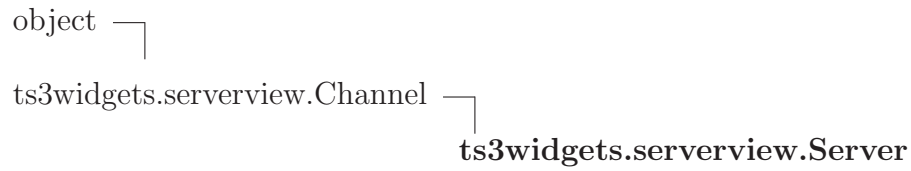
Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

17.2.2 Properties

Name	Description
Inherited from object	
__class__	

17.3 Class Server



Object wrapper for a TS3 server connection.

17.3.1 Methods

__init__(*self*, *schid*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

update(*self*)

Overrides: ts3widgets.serverview.Channel.update

name(*self*)

Overrides: ts3widgets.serverview.Channel.name

iconID(*self*)

Overrides: ts3widgets.serverview.Channel.iconID

rowOf(*self*, *obj*=`None`)

Overrides: ts3widgets.serverview.Channel.rowOf

iconVariable(*self*)

Overrides: ts3widgets.serverview.Channel.iconVariable

Inherited from ts3widgets.serverview.Channel(Section 17.2)

__iter__(), append(), child(), codec(), count(), isDefault(), isFull(), isPasswordProtected(), isPermanent(), isSpacer(), isSubscribed(), maxClients(), neededTalkPower(), remove(), sort(), sortOrder(), spacerAlignment(), spacerCustomtext(), spacerType()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

17.3.2 Properties

Name	Description
Inherited from object	
__class__	

17.4 Class Client

object
└ **ts3widgets.serverview.Client**

Object wrapper for a connected client on a TS3 server.

17.4.1 Methods

__init__(*self*, *schid*, *clid*, *isme*)

x.__init__(...) initializes x; see help(type(x)) for signature

Overrides: object.__init__ extit(inherited documentation)

update(*self*)

count(*self*)

rowOf(*self*)

__**lt**__(*self*, *other*)

__**gt**__(*self*, *other*)

name(*self*)

displayName(*self*)

talkPower(*self*)

isRecording(*self*)

isChannelCommander(*self*)

isTalking(*self*, *val*)

iconID(*self*)

isPrioritySpeaker(*self*)

isAway(*self*)

country(*self*)

isRequestingTalkPower(*self*)

isTalker(*self*)

outputMuted(*self*)

inputMuted(*self*)

hardwareInputMuted(*self*)

hardwareOutputMuted(*self*)

inputDeactivated(*self*)

channelGroup(*self*)

serverGroups(*self*)

iconVariable(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

17.4.2 Properties

Name	Description
Inherited from object	
__class__	

17.5 Class ServerviewModel

PythonQt.QtCore.QAbstractItemModel
└ **ts3widgets.serverview.ServerviewModel**

ItemModel to deliver data of a serverview to ItemWidgets. The data is delivered in one column. Limitations: no badges, no friend/foe status

17.5.1 Methods

__init__(*self*, *schid*, *iconpack*=`None`, *parent*=`None`)

Instantiates a new ServerviewModel object. This raises an exception if the iconpack could not be opened. The object registers itself as callbackproxy to the PythonHost.

Parameters

`schid`: the ID of the serverconnection
(type=int)

`iconpack`: the iconpack to use icons from. defaults to None to use the TS3 client's current IconPack
(type=ts3client.IconPack)

`parent`: the QObject-parent. defaults to None
(type=QObject)

__del__(*self*)

onServerEditedEvent(*self*, *schid*, *editerID*, *editerName*, *editerUID*)

onNewChannelEvent(*self*, *schid*, *cid*, *pcid*)

onNewChannelCreatedEvent(*self*, *schid*, *cid*, *parent*, *invokerID*, *invokerName*, *invokerUniqueIdentifier*)

onUpdateChannelEditedEvent(*self*, *schid*, *cid*, *invokerID*, *invokerName*, *invokerUniqueIdentifier*)

onChannelMoveEvent(*self*, *schid*, *cid*, *newpcid*, *invokerID*, *invokerName*, *invokerUniqueIdentifier*)

onDelChannelEvent(*self*, *schid*, *cid*, *invokerID*, *invokerName*, *invokerUID*)

onClientMoveEvent(*self*, *schid*, *clientID*, *oldChannelID*, *newChannelID*, *visibility*, *moveMessage*)

onChannelUnsubscribeEvent(*self*, *schid*, *channelID*)

onChannelSubscribeEvent(*self*, *schid*, *channelID*)

onClientMoveMovedEvent(*self*, *schid*, *clientID*, *oldChannelID*, *newChannelID*, *visibility*, *moverID*, *moverName*, *moverUniqueIdentifier*, *moveMessage*)

onUpdateClientEvent(*self*, *schid*, *clientID*, *invokerID*, *invokerName*, *invokerUniqueIdentifier*)

onClientSelfVariableUpdateEvent(*self*, *schid*, *flag*, *oldValue*, *newValue*)

onClientMoveSubscriptionEvent(*self*, *schid*, *clientID*, *oldChannelID*, *newChannelID*, *visibility*)

onClientMoveTimeoutEvent(*self*, *schid*, *clientID*, *oldChannelID*, *newChannelID*, *visibility*, *timeoutMessage*)

onClientDisplayNameChanged(*self*, *schid*, *clientID*, *displayName*, *uid*)

onTalkStatusChangeEvent(*self*, *schid*, *status*, *isReceivedWhisper*, *clid*)

onServerGroupListEvent(*self*, *schid*, *serverGroupID*, *name*, *atype*, *iconID*, *saveDB*)

onChannelGroupListEvent(*self*, *schid*, *channelGroupID*, *name*, *atype*, *iconID*, *saveDB*)

index(*self*, *row*, *column*, *parent*)

parent(*self*, *index*)

rowCount(*self*, *parent*)

columnCount(*self*, *parent*)

data(*self*, *index*, *role*)

17.6 Class ServerviewDelegate

PythonQt.QtGui.QStyledItemDelegate
└ **ts3widgets.serverview.ServerviewDelegate**

Delegate to display Serverview items and query the properties and icons from the model to display and show them in one column.

17.6.1 Methods

paint(*self*, *painter*, *option*, *index*)

17.7 Class Serverview

PythonQt.QtGui.QTreeView
└ **ts3widgets.serverview.Serverview**

A QTreeView widget to display the complete view on a TS3 Server.

17.7.1 Methods

__init__(*self*, *schid*, *parent*=`None`)

Instantiates a new Serverview widget (including model and delegate).

Parameters

`schid:` the ID of the serverconnection

(type=int)

`parent:` parent widget

(type=QWidget)

indexToObject(*self*, *index*)

Returns the underlying object of a QModelIndex.

Parameters

`index:` the index of the model

(type=QModelIndex)

Return Value

the wrapped viewitem

(type=Server or Channel or Client)

Index