Python plugin is a server plugin able to run Python scripts that manipulate Crossfire items, monsters, players, maps, ...

The latest version, 2.0, is a major rewrite which introduces Python objects to represent maps, Crossfire objects and players. The hooking system also changed.

Note that this guide doesn't intend to teach you Python. Documentation for that is available on Python's official web site [http://docs.python.org].

Also, advanced functions can require some knowledge of Crossfire's inner workings. Developers seeking help should check Crossfire's page on development [http://crossfire.real-time.com/development/index.html] or Crossfire's resource page [http://crossfire.real-time.com/resources/index.html] (which includes links to lists, IRC, how to contact people, ...). Another option is to check existing Python scripts, available in the maps source tree [http://crossfire.svn.sourceforge.net/viewvc/crossfire/maps/trunk/python/].

Old plugin's reference can be found on plugin_python, for historical purposes.

**Important note:** the API, and other parts, are valid for code's `trunk`. `branch` contains hopefully all the changes too, but there may be differences. Also, links to SVN are to trunk.

# General information

The plugin has access to all built-in functions of Python, and its regular libraries. Python version being used depends on the platform the server was built, but should be 2.3 or higher. Fix Me! {is that always true?}

Using a library not part of Python's default distribution is discouraged as this introduces new dependencies.

Scripts execute in their own context, and their state information is not kept between runs. There are functions to persist data, or share between scripts. Note that data persisted through those functions will not survive a server restart.

Scripts should import the `Crossfire` module to access Crossfire's built-in functions.

Calling Crossfire functions with invalid arguments will result in a Python exception being raised. A function failure will also result in an exception.

Scripts will have the same access rights as the server itself. It may be able to write to some directories, but not others.

Data that should survive a server restart should be persisted in the `PlayerDirectory` (for specific player information) or `DataDirectory` (for regular information), where server normally has write access.

## How do I hook a script to an object?

Python plugin is a server plugin, thus you need to insert in this object one of the *event_xxx* archetype (see `/arch/system` [`http://crossfire.svn.sourceforge.net/viewvc/crossfire/arch/trunk/system/`] directory). The fields to fill are:

- **title**: should be *Python* to call the Python plugin.
- **slaying**: contains the path to the Python script to execute. Root refers to the map directory.

A script can then use `WhoAmI()`, `WhoIsActivator()`, `WhoIsOther()`, `WhatIsMessage()` functions of the `Crossfire` module to find the executing context.

## How do I hook a script to a global event?

Create a script, and put it in the `/python/events/xxx` subdirectory or the `maps` directory, where `xxx` is the event's name in lowercase (create the directory if it doesn't exist). Available events are listed on the server plugin page. In addition, there is a special event, `init`, which is called just after the plugin has been loaded. It behaves like other events in every aspect, but is called only once when the plugin finished loading.

## How do I register a custom command?

Check `Crossfire.RegisterCommand`.

## Return value

The return value of the script can be manipulated through the `GetReturnValue` and `SetReturnValue` functions. Signification of this value depends on the event being handled.

# Python API reference

The following functions and objects are available to all scripts using the Crossfire module through `import Crossfire`.

**Convention:** in the reference, `xxx()` specifies a method, `xxx` specifies a property.

## Constants

Constants are available to wrap values used in Crossfire code like object types, movements, ... They are available through `Crossfire.xxx.VALUE`, where `xxx` is the constant type and `VALUE` the value as in the C source code, in uppercase, without specific prefix (ie F_TRUE becomes TRUE, AT_FIRE becomes FIRE, and so on).

For each constant group `xxx` there exists a `xxxName` dictionary object containing the name of the value as a string.

The following constant types exist:

- `AttackType`: AT_xxx constants, as defined in `include/attack.h` [http://crossfire.svn.sourceforge.net/viewvc/crossfire/server/trunk/include/attack.h?view=markup]
- `AttackTypeNumber`: ATNR_xxx constants, as defined in `include/attack.h` [http://crossfire.svn.sourceforge.net/viewvc/crossfire/server/trunk/include/attack.h?view=markup]
- `CostFlag`: F_xxx constants, as defined in `include/define.h` [http://crossfire.svn.sourceforge.net/viewvc/crossfire/server/trunk/include/define.h?view=markup] (without the F_ prefix)
- `Direction`: contains NORTH, NORTHEAST, ..., NORTHWEST
- `EventType`: EVENT_xxx constants, as defined in `include/plugin.h` [http://crossfire.svn.sourceforge.net/viewvc/crossfire/server/trunk/include/plugin.h?view=markup]
- `MessageFlag`: NDI_xxx constants, as defined in `include/newclient.h` [http://crossfire.svn.sourceforge.net/viewvc/crossfire/server/trunk/include/newclient.h?view=markup]
- `Move`: movement types, as defined in `include/define.h` [http://crossfire.svn.sourceforge.net/viewvc/crossfire/server/trunk/include/define.h?view=markup]
- `Type`: object type, as defined in `include/define.h` [http://crossfire.svn.sourceforge.net/viewvc/crossfire/server/trunk/include/define.h?view=markup]

`MessageFlag` and `Move` can be combined, using +, to build multiple values.

{Fix Me!} do a nice table, with link to values?

Example:

```
whoami.Say("%s => %d"%(Crossfire.DirectionName[Crossfire.Direction.NORTH], Crossfire.Direction.NORTH))
```

will write

```
NORTH => 1
```

**Warning**: Python doesn't know about constants, thus scripts can modify those values. Doing this can lead to weird results, and should be avoided at all cost. Note that the C code itself is immune to the changes, only the scripts would suffer.

## Crossfire module methods

### Event-related functions

Those functions let the script know in what context it was called.

### WhoAmI

Returns the `Crossfire.Object` from which the event being handled was raised, corresponds to the `op` parameter of the event. Can be `None`. For a map event, this is a `Crossfire.Map`

### WhoIsActivator

Returns the `Crossfire.Object` having caused the event being handled, corresponds to the `activator` parameter of the event. Can be `None`.

### WhoIsOther

Returns the `Crossfire.Object` being part of the event, corresponds to the `third` parameter of the event. Can be `None`.

### WhatIsMessage

Returns the `String` that was sent as part of the event, corresponds to the `message` parameter of the event.

### ScriptName

Returns the absolute path of the currently executing script.

### ScriptParameters

This `String` variable contains:

- for a global event, the name of the event (`born`, `gkill`, ...)
- for an event hooked to an object, the `name` field of the object
- for a custom command, the parameters the player sent

### WhatIsEvent

`Crossfire.Object` representing the event object that links to the plugin. Will be `None` for global events and custom commands.

### GetReturnValue

Return current return value.

### SetReturnValue

Parameters:

- return value (integer)

Sets the value the plugin will return to the server when the script exits. Value and effect depend on the event being handled.

### PluginVersion

Returns an integer representing the Python plugin version.

## Configuration-related functions

Those functions let the script know about various server settings.

### MapDirectory

Returns the system directory containing the maps. Note that it is relative to the `Data` directory.

### UniqueDirectory

Returns the system directory where unique items and maps are stored.

### TempDirectory

Returns the system directory where temporary files are stored.

### ConfigDirectory

Returns the system directory containing configuration files.

### LocalDirectory

Returns the system directory containing read/write files.

### PlayerDirectory

Returns the system directory containing player files.

### DataDirectory

Returns the system directory containing read-only data.

## Data access functions

Those functions enable the script to browse runtime data and objects (maps, players, ...).

### FindPlayer

Parameter:

- player name (string)

Returns a `Crossfire.Player` object for specified player, based on partial name matching, or `None` if not found. If only one player name can match, return this player, else return `None`.

### GetArchetypes

Returns a Python `list` containing `Crossfire.Archetype` objects representing all archetypes the server knows. Should not be called lightly, as this can take some time.

### GetMaps

Returns a Python `list` containing `Crossfire.Map` objects, one for each map currently being loaded. This includes random and unique maps.

### GetParties

Returns a Python `list` containing `Crossfire.Party` objects, one for each existing party on the server.

### GetRegions

Returns a Python `list` containing `Crossfire.Region` objects, one for each region.

### GetFriendlyList

Returns a `list` of `Crossfire.Object` representing all items on the friendly list. Individual objects can be added/removed through their `Friendly` flag.

### GetPlayers

Returns a `list` containing `Crossfire.Players` objects, one for each player on the server

### GetTime

Returns a Python `list` corresponding to the in-game server time. Fields are (start at `0` to access):

1. year
2. month
3. day

4. hour
5. minute
6. day of week
7. week of month
8. season

### MapHasBeenLoaded

Parameter:

- map path (`String`)

Returns the `Crossfire.Map` with specified path, or `None` if such a map isn't loaded. Will not try to load the map, see `Crossfire.ReadyMap` for that.

### FindFace

Parameter:

- face name (`String`)

Returns the number of the specified face, 0 if it doesn't exist. The face name is the name as it appears in the archetypes, ie `dcross-red1.111`, without the set name or extension.

### FindAnimation

Parameter:

- animation name (`String`)

Returns the number of the specified animation, 0 if it doesn't exist. The name is has it appears in the archetypes, ie `campfire` or `bat`.

## Action functions

Those functions enable the script to manipulate runtime objects.

### ReadyMap

Loads specified map, and returns a `Crossfire.Map` object wrapping it. Will return `None` if map couldn't be found.

Parameter:

- `String`, the map path to load
- `Number`, flags, optional. Combination of the following values:
  - 2: map is a player's unique map. Path is assumed to be the full path already, and it won't be changed
  - 16: map is reloaded anyway, even if it is already loaded

### CreateObject

Returns an empty object which can then be manipulated and inserted in a map.

### CreateObjectByName

Parameters:

- object or archetype name (`String`)

Returns object with specified archetype or object name, or `None` if no object has the name, and archetype wasn't found.

### CreateMap

Parameters:

- map width (`Number`)
- map height (`Number`)

Returns an empty map of specified dimensions. Unless map is removed immediately, `Path` should be set so the map correctly displays in the `maps` output.

## Misc functions

### RegisterCommand

Parameters:

- command name (string)
- script (string)
- speed (decimal, must be positive or 0)

Registers a server command which will be mapped to specified script. See Registering a command for more information.

The script path is relative to the `maps` subdirectory. When the command is called, the following methods will provide information:

- `WhoAmI` contains the player
- `ScriptName` contains the currently executing script's name
- `ScriptParameters` contains any parameter the player added to the command.

{can NULL be passed?}

### RegisterGlobalEvent

Parameters:

- event code (`Number`, should be one of the `Crossfire.EventType.xxx` constants)

This instructs the server to call the Python plugin for all events of specified type. Python will then call scripts in the `/python/events/xxx` directory, where `xx` is the event's name.

This function should not be used except in special circumstances, as Python by default registers all events anyway.

### UnregisterGlobalEvent

Parameters:

- event code (`Number`, should be one of the `Crossfire.EventType.xxx` constants)

This instructs the server to stop calling the Python plugin for all events of specified type.

This function should not be used, except in special circumstances, as it can disable other scripts that use those events.

### DestroyTimer

Removes a timer created through a call to `Crossfire.Object.AddTimer`.

Arguments:

- `Number`: timer identifier.

Returns:

- 0: no error.
- -1: invalid timer identifier.

### GetPrivateDictionary

Returns a Python `dictionary` the script can use to store data between runs. This data is not shared with other scripts.

Note that data will be lost in case of server restart.

### GetSharedDictionary

Returns a Python `dictionary` the script can use to share data with other scripts.

Note that data will be lost in case of server restart.

### MatchString

Parameters:

- string to be searched
- pattern to search for

Returns `1` if string matches the pattern, `0` else.

Pattern can be a regular expression Fix Me! specifications

## Crossfire.Archetype methods and attributes

This represents an archetype, from which all objects are derived. Pointer to such item can be obtained through the `Crossfire.Object.Archetype` or the list returned by `Crossfire.GetArchetypes()`.

Properties:

- Clone: returns a `Crossfire.Object` representing the default values for items of that archetype. Will never be `None`. Its values can't be changed (but trying to change one will not result in any error)
- Head: archetype to which this archetype is linked to. Will be `None` if not applicable
- Name: archetype's name
- Next: next archetype in the archetype list. Will be `None` for last item
- More: next archetype linked to current archetype. Will be `None` for last item

Methods:

- NewObject: returns a `Crossfire.Object` having this archetype as type. It isn't on any map or in any inventory, so should be inserted somewhere or deleted if not used.

## Crossfire.Object methods and attributes

Fix Me! link to dev:objects 's fields? split flag & such info. Make a table with property / mapping to object / Python type Properties in **bold** are read-write, others are readonly. Most properties are mapped to fields of the `object` structure. `Boolean` values are mapped to the various flags.

- **Name**: `String` containing the object's name
- **NamePl**: `String` containing the object's plural name
- **Title**: `String`
- **Race**: `String`
- **Map**: `Crossfire.Map` containing the map the object is in. Changing that teleports the object to specified map, at default coordinates.
- **Cha**: charisma
- **Con**: constitution
- **Dex**: dexterity
- **Int**: intelligence
- **Pow**: power
- **Str**: strength
- **Wis**: wisdom
- **HP**: hitpoints
- **MaxHP**: maximum hitpoints
- **SP**: spellpoints
- **MaxSP**: maximum spellpoints
- **Grace**: grace
- **MaxGrace**: maximum grace
- **Food**: food
- **AC**: armour class
- **WC**: weapon class
- **Dam**
- Luck
- **Exp**
- PermExp
- ExpMul
- **Message**: `String`
- **Slaying**: `String`
- **Cursed**: `Boolean`

- **Damned**: `Boolean`
- **Weight**: `Number`
- **WeightLimit**: `Number`
- Above: `Crossfire.Object`
- Below: `Crossfire.Object`
- Inventory: `Crossfire.Object`
- X: `Number`
- Y: `Number`
- **Direction**: `Number`
- **Facing**: `Number`
- **Unaggressive**: `Boolean`
- **God**: `String`
- **Pickable**: `Boolean`, whether the item can be picked up or not
- **Quantity**: `Number`. Setting it to 0 removes the object.
- **Invisible**: `Boolean`
- **Speed**: `Number`
- **SpeedLeft**: `Number`
- **LastSP**: `Number`
- **LastGrace**: `Number`
- Level: `Number`
- **Face**: when reading, `Number` representing the face number. When writing, a `String` representing the face's name (eg: `cobblesto1.111`). Will raise an error if invalid face.
- **Anim**: when reading, `Number` representing the animation number, 0 if none. When writing, a `String` representing the animation's name (eg: `fireplace`). Will raise an error if invalid animation.
- **AnimSpeed**: `Number`, the animation speed
- **AttackType**: `Number`
- BeenApplied: `Boolean`
- Identified: `Boolean`
- Alive: `Boolean`
- DungeonMaster: `Boolean`
- WasDungeonMaster: `Boolean`
- **Applied**: `Boolean`
- **Unpaid**: `Boolean`
- Monster: `Boolean`
- **Friendly**: `Boolean`
- Generator: `Boolean`
- Thrown: `Boolean`
- **CanSeeInvisible**: `Boolean`
- **Rollable**: `Boolean`
- **Turnable**: `Boolean`
- **UsedUp**: `Boolean`
- Splitting: `Boolean`
- **Blind**: `Boolean`
- CanUseHorn: `Boolean`
- CanUseRod: `Boolean`
- CanUseSkill: `Boolean`
- **KnownCursed**: `Boolean`
- **Stealthy**: `Boolean`
- **Confused**: `Boolean`
- **Sleeping**: `Boolean`
- **Lifesaver**: `Boolean`
- Floor: `Boolean`
- HasXRays: `Boolean`
- CanUseRing: `Boolean`
- CanUseBow: `Boolean`
- CanUseWand: `Boolean`
- **CanSeeInDark**: `Boolean`
- **KnownMagical**: `Boolean`
- CanUseWeapon: `Boolean`
- CanUseArmour: `Boolean`
- CanUseScroll: `Boolean`
- CanCastSpell: `Boolean`
- **ReflectSpells**: `Boolean`
- **ReflectMissiles**: `Boolean`

- **Unique**: `Boolean`
- CanPickUp: `Boolean`, whether the monster can pickup items or not.
- **RunAway**: `Boolean`
- **Scared**: `Boolean`
- **Undead**: `Boolean`
- **BlocksView**: `Boolean`
- **HitBack**: `Boolean`
- **StandStill**: `Boolean`
- **OnlyAttack**: `Boolean`
- **MakeInvisible**: `Boolean`
- Money: `Number`, returns the total money (in silver) the object owns
- Type: `Number`
- Subtype: `Number`
- **Value**: `Number`, value of the object
- ArchName: `String`, name of this object's archetype
- Archetype: `Crossfire.Archetype`
- **NoSave**: `Boolean`
- Exists (special meaning, False if cf object has been freed from memory, True otherwise)
- Env: `Crossfire.Object`. Can be `None`.
- MoveType
- MoveBlock
- MoveAllow
- MoveOn
- MoveOff
- MoveSlow
- MoveSlowPenalty
- **Enemy**: `Crossfire.Object` representing the object's enemy. Will probably be `None` except for monsters.
- Count
- **GodGiven**: `Boolean`
- **IsPet**: if the object is on the friendly list or not
- **AttackMovement**
- **Duration**
- **Skill**

## Methods

Fix Me! document all parameters. Link to relevant Crossfire function. Split in "standard function" and "helper function"?

### ActivateRune

Current object springs like a trap, hitting the victim.

Arguments:

- `Crossfire.Object` that is the victim of the trap.

### AddExp

Adds or substracts experience to current object.

Arguments:

- `number`, experience to add
- `string` (optional), skill to add to
- `number` (optional), what to do if player doesn't have the specified skill:
  - 0: give the player the skill
  - 1: give player exp to total, no skill
  - 2: player gets nothing
  - 3: used when removing exp

### Apply

Current object applies specified object. The effect depends on the object being applied, which can be a potion, a trigger, ...

Arguments:

- `Crossfire.Object` to apply
- flag meaning?

Return value:

- 0: player or monster can't apply objects of that type
- 1: has been applied, or there was an error applying the object
- 2: objects of that type can't be applied if not in inventory

### Cast

Current object casts a spell.

Arguments:

- `Crossfire.Object` representing the spell to case
- `Crossfire.Direction`, direction in which the object should cast
- `String`, optional arguments to the spell (for food creation, ...)

Return value:

- 0 if spell wasn't cast
- non-zero value if spell was cast

Note that all checks related to spell-casting apply: enough sp/gp, knowing the skill, sufficient level, spot where magic isn't forbidden, ...

The spell needn't be known to be known to be cast, though.

### CastAbility

Equivalent of `Cast`.

### ChangeAbil

Displays relevant messages about resistances and so on changes.

Arguments:

- `Crossfire.Object` which is a force effect, or something that changes a player's attribute and was just added to inventory.

### CheckArchInventory

Finds an object in current object's inventory.

Arguments:

- `String` representing the object name to find

Will return the first object in inventory that has the specified name. Name must be the raw name, without bonuses and such.

### CheckInventory

Finds an object in current object's inventory.

Arguments:

- `String` representing the object to find

Will return the first object in inventory that has the specified name. Matching is done on archetype, and also match on start of the object name.

### CheckTrigger

Checks if current object should/can react to another object moving on it.

Arguments:

- `Crossfire.Object`, `cause` that can cause current Object to react (altar, ...)

Return value:

- for TRIGGER_ALTAR objects, 1 if `cause` was destroyed, 0 if not.
- TRIGGER: 1 if handle could be moved, 0 if not.
- TRIGGER_BUTTON, TRIGGER_PEDESTAL: 0.

### CreateObject

Create an object inside the current object.

Arguments:

- `String` which is the name of the object to create

Will create a new `Crossfire.Object` from specified name (like `Crossfire.CreateObjectByName`) and insert it into the current object.

### CreateTimer

Fires an `event_timer` after the specified delay. Object should have a handler for this event.

Arguments:

- `number`, delay for timer
- `number`, timer mode
  - 1: delay is in seconds
  - 2: delay is in server cycles

Returns: timer identifier, which can be used to remove the timer through a call to `Crossfire.DestroyTimer`

### Drop

Current object drops specified object, which will be put on the ground, or in a container, depending on applied containers.

Argument:

- `Crossfire.Object` to drop

### Fix

Current object is reinitialized from its default values, values (ac, wc, Str, ...) are recomputed from items worn, in inventory, ...

### ForgetSpell

Player forgets a spell.

Arguments:

- `Crossfire.Object` representing the spell to forget

Object forgetting should be a player.

### GetResist

Gets the value of a resistance to an attacktype.

Arguments:

- `Number`: attacktype for which to get the resistance

Return: `Number` with the value. Note that an invalid attacktype will return 0, which can't be distinguished from a neutral resistance.

### InsertInto

Insert the current object into another object.

Arguments:

- `Crossfire.Object`, in which the current object will be inserted

Return:

- `Crossfire.Object` representing the newly inserted object.

After using this function, you should not use the initial `Crossfire.Object`, as it can be merged with others.

### KnowSpell

Checks if current object knows a spell.

Arguments:

- `String`, name of the spell to check for

Return:

- `Crossfire.Object` representing the spell the object known, or `None` if this spell isn't known to the currect `Crossfire.Object`.

### LearnSpell

Make the current object learn a spell.

Arguments:

- `Crossfire.Object` representing the spell to learn

There is no learning failure. Note that any spell can be learned this way. Player will receive the message that the spell was learnt correctly.

### Move

Tries to move the object in a direction.

Arguments:

- `Number`, direction to move to. Should be one of `Crossfire.Direction.xxx` value.

Return:

- 1 if object could move
- 0 else

### OutOfMap

Check a position on the object's map.

Arguments:

- `Number`, `x` and `y` coordinates

Return:

- 1 if coordinates are in the map the current object is on
- 0 else

Note that this function takes into account maps tiling.

### Pay

Buys an item.

Arguments:

- `Crossfire.Object` to buy

Return value:

- 1 if item could be bought
- 0 else

This function applies bargaining skill if applicable, and will handle shop-based price difference. Note that the object to buy doesn't need to have the FLAG_UNPAID set.

### PayAmount

Pays money.

Arguments:

- `Number`, amount to pay for, in raw value units (silver coin)

Return:

- 1 if object could pay specified amount
- 0 else

### QueryCost

Finds the price of an object.

Arguments:

- `Crossfire.Object` that current `Crossfire.Object` wants to know the price of
- `Number`, flags to apply, combination of `Crossfire.CostFlag` values

Return value:

- price of the item

### QueryName

Return: `String` containing the object's full name.

### ReadKey

Reads key associated to value.

Arguments:

- `String`: key value to read

Return: `String` containing the value. Will be empty if NULL value.

### Remove

Destroys current object, which then becomes invalid.

### Reposition

Moves current object to specified position on its current map.

Arguments:

- `Number`, `x` and `y`, coordinates of position to teleport the object to

### Say

Current object says something in current map.

Argument:

- `string` to say.

### Speak

Current object says something in current map

Argument:

- `string` to say.

Equivalent to `Say`.

### Take

Current object picks up specified object, which will be put in inventory, or in a container, depending on applied containers.

Argument:

- `Crossfire.Object` to take

### Teleport

Teleports the object to specified place.

Arguments:

- `Crossfire.Map`: map to insert into.
- `Number`: x coordinates
- `Number`: y coordinates

Return value: `Number`, 0 for success, 1 for failure.

### WriteKey

Inserts specified key/value in the object.

Arguments:

- `String`: key value
- `String`: value to associate to the key
- `Number` (optional): 0 to only update the key and not insert it, non zero to add it anyway

## Crossfire.Map methods and attributes

Properties in **bold** are read-write, others read only.

- Difficulty: `Number`
- **Path**: `String`
- TempName: `String`
- Name: `String`
- ResetTime: `Number`
- ResetTimeout: `Number`
- Players: `Number`, number of players on the map
- Light: `Number`
- Darkness: `Number`
- Width: `Number`
- Height: `Number`
- EnterX: `Number`
- EnterY: `Number`
- Temperature: `Number`
- Pressure: `Number`
- Humidity: `Number`
- WindSpeed: `Number`
- WindDirection: `Number`
- Sky: `Number`
- WPartX: `Number`
- WPartY: `Number`
- Message: `String`
- Region: `Crossfire.Region`
- Exists (special meaning, `False` if cf map has been freed from memory, `True` otherwise)
- Unique: `Number`

### Print

Prints the specified message to all players on the map.

Arguments: * `String`, message to print * `Number`, optional flags, combination of `Crossfire.MessageFlag` (default is NDI_BLUE|NDI_UNIQUE).

### ObjectAt

Returns the first `Crossfire.Object` at specified location.

Arguments:

- `Number`, `x` and `y`

Other objects on the position are available through the use of the `Above` field.

### CreateObject

Creates an object on the map.

Arguments:

- `String`, name of object to create
- `Integer`, `x` and `y`, coordinates

Equivalent of calling `Crossfire.CreateObjectByName()` (creates an item from archetype or name) and inserting it in the map at the specified location

### Check

Arguments:

- `String`, name of object to check for. Should be a raw name, without the bonuses and title and such.
- `Integer`, optional `x` and `y` coordinates

Return:

- the first object at the specified location matching specified name

### ChangeLight

Change the map's light level.

Arguments:

- `Number`, specifying how to change the light. Negative value means map gets brighter, positive darker.

Note that light will never be negative or over 5 (`MAX_DARKNESS`)

### Insert

Inserts an object at specified location.

Arguments:

- `Crossfire.Object` to insert
- `Number`, X and Y, coordinates where to insert

Return: `Crossfire.Object` representing the inserted object. Can be `None` if object was removed for some reason (eaten by an altar, ...)

## Crossfire.Player methods and attributes

This class inherits from Crossfire.Object, and introduces the new properties (bold = read-write):

- `IP`: `String`.
- **BedMap**: `string` containing the map in which the player last saved.
- **BedX** : `number` containing the x coordinate of last savebed.
- **BedY** : `number` containing the y coordinate of last savebed.
- **MarkedItem**: `Crossfire.Object`, item the player has marked.
- **Party**: `Crossfire.Party`, party in which the player is. Note that changing it bypasses a potential party's password.

### CanPay

Checks if player can pay for unpaid items in inventory.

Return value:

- 1 if can pay

- 0 else

This will print suitable messages if player can't pay.

### Message

Sends a message to the player.

Arguments:

- `String`, message to say
- `Number`, optional, combination of `Crossfire.MessageFlag`, default value is `NDI_UNIQUE + NDI_ORANGE`.

### Write

Equivalent of Message.

## Crossfire.Party methods and attributes

This class merely encapsulates a party. Everything is read-only. Attributes:

- Name: `String`
- Password: `String`
- Next: `Crossfire.Party`

### GetPlayers

Return value: a `list` of `Crossfire.Players`

## Crossfire.Region methods and attributes

This class encapsulates an ingame region. Properties are read-only.

- Name: `String`
- Longname: `String`
- Message: `String`
- Next: `Crossfire.Region`

### Parent

Return value: a `Crossfire.Region`, or `None` if no parent region.

# Scripts part of Crossfire default maps

Those Python scripts reside in the python [http://crossfire.svn.sourceforge.net/viewvc/crossfire/maps/trunk/python/] subdirectory of the maps tree. They provide additional functionality, and may be considered as always present. Fix Me! {describe existing ones}

## Dialog Helper

Script location: `/python/CFDialog.py`

This script provides two utility classes a map maker can use to easily create complex dialogs. Go here for a complete explanation on how it works.

## Death message

Script location: `/python/misc/death_message.py`

This script makes the monster/living thing containing it display a message when it dies. It should be hooked through an `event_death` archetype in monster's inventory.

Message is specified in event object's options field. It can contain %m and %k, which will be replaced by respectively dead item's name and killer's name (if no killer, replaced by an empty string).

## Greeting message

Script location: `/python/misc/greet_message.py`

This script makes the monster/living thing containing it display a message when it attacks for the first time an enemy. It should be hooked through an `event_time` archetype in monster's inventory.

Message is specified in event object's options field. It can contain %m and %e, which will be replaced by respectively monster's name and enemy's name.

## Experience rewarder

Script location: `/python/items/experience_rewarder.py`

This script will give experience to the "activating" player. It should be linked through the "trigger" event of an altar, or similar thing.

Options are specified in the event object's fields.

A key/value will be used to store in the player if she activated the item already.

Available options are:

- exp: experience to gain
- skill: skill to add experience to. Can be empty, in which case exp is given to the general exp only
- wc: what to do when the player doesn't know the skill:
  - 0: give the player the skill
  - 1: give player exp to total, no skill
  - 2: player gets nothing
- race: if set, the player can only use once this item.