I. Introduction

I.a Purpose of CPPI

The purpose of CPPI is to define a standard interface for prop protectors to use for easy third-party integration with any prop protector.

I.b Terms and Definitions

- *Required* – This feature or function is required for the prop protector be considered to have a valid CPPI interface.
- *Recommended* – This feature or function is strongly suggested to be implemented in any CPPI interface but not mandatory. If a function is marked recommended and the CPPI interface does not implement it, it *must* return CPPI_NOTIMPLEMENTED (see below).

I.c Revisions

- 1 November 2014 – Version 1.3 of standard

  o Added CanDrive, CanProperty, and CanEditVariable

  o Changed date formats in this document to something more universally understandable

- 6 August 2010 – Proposed Version 1.2 of standard

  o Changed hook recommendation from gamemode.Call to hook.Call

  o Added third parameter uid to CPPIAssignOwnership, changed definition so return value of true indicates that ownership *should* be assigned
  o Added new required entity functions, CPPICanUse and CPPICanDamage
  o Added ability to clear ownership by passing nil to CPPISetOwner and CPPISetOwnerUID
- 14 June 2008 – Version 1.1 of standard
- 3 May 2008 – Version 1.0 of standard

II. Implementation

II.a Variables

Namespace: CPPI (store in the global table 'CPPI')

Variable: **CPPI_DEFER --** *Required*
- Can be any value you please, just make sure it doesn't conflict with other possible return values from functions it's used in.
- This variable is used for certain functions that allow its usage to tell the calling function that it doesn't know the value requested yet.
- This is useful for client side functions that need to get a variable from the server.

Variable: **CPPI_NOTIMPLEMENTED –** *Required*
- Can be any value you please, just make sure it doesn't conflict with other possible return values from functions it's used in.
- This variable is used for certain function that allow its usage to tell the calling function that the feature requested is not implemented by this prop protection.
- For example, a prop protector that does not implement client side ownership would return this for the client side ownership functions.

II.b Global Functions

Namespace: CPPI (store in the global table 'CPPI')

Function: **GetName() –** *Required*
- Returns the name of the active prop protection (IE, "Amazing Prop Sprite")
- Return value constraints: String of less than 255 characters
- Both client and server side.

Function: **GetVersion() –** *Required*

- Returns the version of the active prop protection (IE, "1.0" or "1.5.6a rev258")
- Return value constraints: String of less than 255 characters
- Both client and server side.

Function: **GetInterfaceVersion() –** *Required*

- Returns the version of the interface (the version of *this* standard that the interface is using)
- Return value constraints: returns the Number 1.3.
- Both client and server side.

Function: **GetNameFromUID(** *[String]* **uid) –** *Recommended*
- Returns the name of the player that was using that UID (Player:UniqueID())
- uid will be a number specifying a UID (but do error checking regardless)
- Return value constraints: String of player name less than 32 characters, or nil if a match cannot be found.
- Both client and server side, if the function is implemented client-side it is only required to function on the local player's UID client-side.

- This function is allowed to return *CPPI_DEFER*.

II.c Player Functions

Namespace: Player (the player metatable)
Function: **CPPIGetFriends() –** *Recommended*
- Returns the friends of a player, other players who are allowed to manipulate the specified player's objects.
- Ply argument: A valid Player object.
- Return value constraints: Must be a table of less than or equal to 64 Player elements.
- If this is implemented, it is recommended to be both client and server side, but it is perfectly acceptable for the client side is to only implement this function for the local player. Return *CPPI_NOTIMPLEMENTED* on the client if you have not implemented the function for anything other than the local player if you receive a non-local player Player object.

- This function is allowed to return CPPI_DEFER.

II.d Entity Functions

Namespace: Entity (the entity metatable)

Function: **CPPIGetOwner() –** *Required*
- Returns the Player object that owns this entity first.
- Returns the UID (Player:UniqueID()) of the owner second.
- On the return values, both values should be nil if no one owns it, first value should be nil if the player is disconnected. If the prop protection does not keep track of UIDs, the second value should be CPPI_NOTIMPLEMENTED.
- The client version of this function can always return CPPI_NOTIMPLEMENTED if the prop protection does not track owners on the client.

- This function is allowed to return *CPPI_DEFER* (for both values).

Function: **CPPISetOwner(** *[Player]* **ply) –** *Required*

- Sets a new owner for this entity.
- Returns true on success, false on failure.
- ply will be a Player entity or nil (do error checking regardless).
- If ply is nil, ownership should be cleared for the entity.
- ONLY defined on the server.

Function: **CPPISetOwnerUID(** *[String]* **uid) –** *Recommended*

- Sets a new owner for this entity.
- Returns true on success, false on failure, or CPPI_NOTIMPLEMENTED.
- uid will be a uid for a player or nil (do error checking regardless).
- If uid is nil, ownership should be cleared for the entity.
- ONLY defined on the server.

Function: **CPPICanTool(** *[Player]* **ply,** *[String]* **toolmode) –** *Required*

- Checks if a player can tool this object with specified tool. This function *MUST NOT* reassign ownership; that the player will actually use the tool is not guaranteed.
- Returns true if they can, false if they can't.
- ply will be a Player entity (do error checking regardless).
- toolmode will be a String specifying the tool mode to check. Not guaranteed to be one of the default Garrysmod tools.
- *Hint: This should be a close alias of your CanTool hook .*
- ONLY defined on the server.

Function: **CPPICanPhysgun(** *[Player]* **ply) –** *Required*

- Checks if a player can physgun this object. This function *MUST NOT* reassign ownership; that the player will actually use the physgun is not guaranteed.
- Returns true if they can, false if they can't.
- ply will be a Player entity (do error checking regardless).
- *Hint: This should be a close alias of your PhysgunPickup hook .*
- ONLY defined on the server.

Function: **CPPICanPickup(** *[Player]* **ply) –** *Required*

- Checks if a player can pick up this object with the gravgun. This function *MUST NOT* reassign ownership; that the player will actually pick up the object is not guaranteed.
- Returns true if they can, false if they can't.
- ply will be a Player entity (do error checking regardless).
- *Hint: This should be a close alias of your GravGunPickupAllowed hook.*
- ONLY defined on the server.

Function: **CPPICanPunt(** *[Player]* **ply) –** *Required*
- Checks if a player can punt this object with the gravgun. This function *MUST NOT* reassign ownership; that the player will actually punt the object is not guaranteed.
- Returns true if they can, false if they can't.
- ply will be a Player entity (do error checking regardless).
- *Hint: This should be a close alias of your GravGunPunt hook.*
- ONLY defined on the server.

Function: **CPPICanUse(** *[Player]* **ply) –** *Required*
- Checks if a player can use this object (usually press 'e' on it). This function *MUST NOT* reassign ownership; that the player will actually use the object is not guaranteed.
- Returns true if they can, false if they can't.
- ply will be a Player entity (do error checking regardless).
- *Hint: This should be a close alias of your PlayerUse hook.*
- ONLY defined on the server.

Function: **CPPICanDamage(** *[Player]* **ply) –** *Required*
- Checks if a player can damage this object. This function *MUST NOT* reassign ownership; that the player will actually damage the object is not guaranteed.
- Returns true if they can, false if they can't.
- ply will be a Player entity (do error checking regardless).
- *Hint: This should be similar to your EntityTakeDamage hook.*
- ONLY defined on the server.

Function: **CPPIDrive(** *[Player]* **ply) –** *Required*
- Checks if a player can drive this object. This function *MUST NOT* reassign ownership; that the player will actually drive the object is not guaranteed.
- Returns true if they can, false if they can't.
- ply will be a Player entity (do error checking regardless).
- *Hint: This should be a close alias of your CanDrive hook.*
- ONLY defined on the server.

Function: **CPPICanProperty(** *[Player]* **ply,** *[String]* **property) –** *Required*
- Checks if a player can use a property on this object. This function *MUST NOT* reassign ownership; that the player will actually use the property on the object is not guaranteed.
- Returns true if they can, false if they can't.
- ply will be a Player entity (do error checking regardless).
- property will be a String specifying the property to check.
- *Hint: This should be similar to your CanProperty hook.*
- ONLY defined on the server.

Function: **CPPICanEditVariable(** *[Player]* **ply,** *[String]* **key,** *[String]* **val,** *[Table]* **edit)** – *Required*
- Checks if a player can edit a variable on this object. This function ***MUST NOT*** reassign ownership; that the player will actually change the variable on the object is not guaranteed.
- Returns true if they can, false if they can't.
- ply will be a Player entity (do error checking regardless).
- key will be a String specifying the key to check.
- val will be a String specifying the value to check.
- edit will be a Table specifying the edit table to check.
- *Hint: This should be similar to your CanEditVariable hook.*
- ONLY defined on the server.

II.e Hook Functions

Namespace: Hooks (normal gamemode hooks called using hook.Call() and hooked using hook.Add())

Function: **CPPIAssignOwnership(** *[Player]* **ply,** *[Entity]* **ent,** *[String]* **uid)** – *Recommended*
- Called when ply assumes ownership of ent.
- If the hook is blocked by returning false, ownership must not be assigned.
- If the hook is blocked by returning any value other than false or nil, ownership should be assigned.
- ply should be a valid Player entity if the player is connected.
- ent should be a valid Entity that the ply now owns.
- uid should be a the String of the owner's UID.
- If ownership is being cleared, the ply and uid arguments should both be nil.
- *Hint: This should be called both after a player spawns a new ent and when ownership is reassigned.*
- Only recommended on the server, but preferred on client as well.

Function: **CPPIFriendsChanged(** *[Player]* **ply,** *[Table]* **newfrends)** – *Recommended*

- Called when ply gains or loses friends (other players who are allowed access to their ents).
- Should be called after friends are re-assigned, cannot be blocked.
- ply should be a valid player entity that had their friends changed
- newfriends should be a table containing the player's friends.
- *Hint: This should be called both after a player spawns in the server and when their friends change*
- Only recommended on the server, but preferred on client as well (at least for local player).