Mission

Code

```
enum                                                                    MissionState
{
 MISSION_RUNNING, //                                läuft
 MISSION_SUCCEEDED, //                             gewonnen
 MISSION_FAILED  //                               verloren
};
```

Code

```
enum                                                            ActionCallbackResult
{
 ACTION_CONTINUE,
 ACTION_SKIP,
 ACTION_CLEAR
};
```

Code

```
enum                                                              PathFinishedAction
{
 PATH_DEFAULT,
 PATH_DELETE,
 PATH_RESTART,
 PATH_STOP
};
```

Code

```
enum                                                                  MoveCollCheck
{
 MCC_HALT_CONTINUE,    // default: on collision stop moving the object, but don't stop action
 MCC_IGNORE_CONTINUE,  //   ignore   collision   (move   on)   and   continue   action
 MCC_HALT_DONE,    //       stop       moving       and       remove       action
 MCC_IGNORE_DONE,   //  continue  moving  (for  this  step)  and  remove  action
 MCC_HALT_CLEARLIST,   //     stop     moving     and     clear     action     list
 MCC_IGNORE_CLEARLIST  // continue moving (for this step) and clear action list
};
```

Code

```
namespace Mission
{
 struct MissionScoring
 {
  float PersonResult;
  float ThingsResult;
  float BudgetResult;
  float TimeResult;
  float Efficiency;
 };

 void AddObjective(const char *Name_, const char *OpenComment_ = NULL, const char *AccomplishedComment_ = NULL);
 void AddObjectivePlayComment(const char *Name_, const char *OpenComment_ = NULL, const char *AccomplishedComment_ = NULL);
 void RemoveObjective(const char *Name_);
 void RemoveAllObjectives();
 bool HasObjective(const char *Name_);
 void SetObjectiveAccomplished(const char *Name_, bool State_);
 bool IsObjectiveAccomplished(const char *Name_);
 bool AllObjectivesAccomplished();
 int GetNumObjectivesAccomplished();
 int GetNumObjectivesNotAccomplished();

 bool IsDefaultLogicPositive();
 bool IsDefaultLogicNegative();

 bool StartSingleTimer(const char *Name_, float TimeOut_, bool usePhysicsTicks_ = false);
 bool StartIntervalTimer(const char *Name_, float TimeOut_, bool usePhysicsTicks_ = false);
 bool StopTimer(const char *Name_);
 bool PauseTimer(const char *Name_);
 bool ResumeTimer(const char *Name_);
 float ModifyTimeOut(const char *TimerName_, float TimeDifference_);
 bool TimerIsStarted(const char* name);

 int GetCounter(const char *Counter_);
 int IncCounter(const char *Counter_, int Amount_=1);
 int DecCounter(const char *Counter_, int Amount_=1);
 int SetCounter(const char *Counter_, int Value_);
 void ResetCounters();

 void StartCutScene(bool useCamLimitsForTransition_ = true);
 void EndCutScene(bool restoreOldCamLocation_ = false, float restoreTransitionDuration_ = 0.0f);
 bool IsCutSceneRunning();
 void ShowBlackBars(float duration_ = 0.0f);
 void HideBlackBars(float duration_ = 0.0f);
 void CloseBlackBars(float duration_ = 0.0f, float blackDuration_ = 3.0f);

 void PlayHint(const char *HintText, const char *SoundFile = NULL);
 void ShowMessageTickerText(const char *Text_, float Red_=1.0f, float Green_=1.0f, float Blue_=1.0f);
 void PlayComment(const char *commentFile_);
 bool IsCommentPlaying();
 int GetMissionTime();
 int GetMoneyLeft();
 void SetMoney(int Amount_);
 const char *GetDefaultCommentForEfficiency(float efficiency_);

 void StartCountDown(int Seconds_);
 void PauseCountDown();
 void ResumeCountDown();
 void StopCountDown();
 int GetCountDownSeconds();
 void SetCountDownColor(float red_, float green_, float blue_);
};
```

Alles anzeigen

Code

```
class                                                                MissionScript
{
  MissionScript(const      char      *Class_,      const      char      *Object_);
  virtual                                                          ~MissionScript();
};
```