```
Information to document about each primitive:

Primitives number: (enumerated type id)
Implementation file: (file name)
Control panel: (picture)
Parameters: (c structure)
Results: (text)


@       Idle

class IdleParam {
public:
        StdPrm          decStackVar;    // which temporary to decrement
};

@       GenericSimCall

class GenericSimCallParam {
public:
        enum {
                kGeneralTutorialComplete,
                kCenterViewOnStackObject,
                kSetActionIconToStackObject,
                kUncenterViewOnStackObject,
                kAddToFamily,
                kCombineAssets,
                kRemoveFromFamily,
                kMakeNewNeighbor,
                kFamilyTutorialComplete,
                kArchitectureTutorialComplete,
                kDisableModeSwitch,
                kEnableModeSwitch,
                kGetDistanceToCameraInTemp0,
                kAbortInteractions,
                kSetHouseRadioStation,
                kSetRoutingFootprint,
                kSelectNewNormalSuit,
        };
        StdPrm call;
};

@       Expression

class ExpressionParam {
public:
        StdPrm  lhsData,                // the lhs attribute/data/temp/simglob...
number
                        rhsData;                // the rhs
attr/data/temp/immediate... number
        SInt8   isSigned,               // align to 8 bits
                        opType,         // <,>,+= etc
                        lhsOwner,               // the owner of the lhs data
reference
                        rhsOwner;               // the owner of the rhs data
reference
};

@       FindBestAction

class FindBestActionParam {
public:
};
```

```
@       Grab

class GrabParam {
public:
};

@       Drop

class DropParam {
public:
};

@       ChangeSuit

class ChangeSuitParam {
public:
        UInt8 suitIndex;
        UInt8 suitLocation;
        StdPrm flags;

        bool GetUndress() const {return (flags&1)?true:false;}
        void SetUndress(bool undress) {flags&=~1; if (undress) flags|=1;}

        // Possible values for suit location.
        enum {
                kGlobal, // global accessories.
                kPerson, // person suits like bathing, nude
                kObject  // object accessores.
        };
};

@       Update

class UpdateParam {
public:
        StdPrm          who;            // which object to update(owner type)
        StdPrm          what;           // what to update (0=graphic, 1=lighting
contribution)
};

@       Random

class RandomParam {
public:
        StdPrm          destData,               // where to assign
                        destOwner,              // owner of assign
                        rangeData,              // dynamic range
                        rangeOwner;             // range owner
};

@       Burn

class BurnParam {
public:
        UInt8 what;
        enum {
                kStackObj,
                kTileInFronOfStackObj,
                kFloorUnderStackObj
        };
        UInt8 flags;

        bool GetAllowBusyObjects() const {return (flags&1)?true:false;}
        void SetAllowBusyObjects(bool allow) {flags&=~1; if (allow) flags|=1;}
};
```

```
@       Tutorial

class TutorialParam {
public:
        UInt8 action;

        enum {
                kBegin,
                kEnd
        };
};

@       DistanceTo

class DistanceToParam {
public:
        StdPrm          destTemp; // which temp to assign gotten distance to
        UInt8           flags;
        UInt8           fromOwner;
        StdPrm          fromData;

        StdPrm GetFromOwner() const { if ((flags&1)==0) return 3; return
fromOwner; } // "3" is kMyData from XOwners.h
        void SetFromOwner(StdPrm owner) { flags |= 1; fromOwner = UInt8(owner); }

        StdPrm GetFromData() const { if ((flags&1)==0) return 11; return fromData;
} // "11" is kObjectID field of objects
        void SetFromData(StdPrm data) { flags |= 1; fromData = data; }
};

@       DirectionTo

class DirectionToParam {
public:
        StdPrm          destData,       // which data field to assign to
                                destOwner;
        UInt8           flags;
        UInt8           fromOwner;
        StdPrm          fromData;

        StdPrm GetFromOwner() const { if ((flags&1)==0) return 3; return
fromOwner; } // "3" is kMyData from XOwners.h
        void SetFromOwner(StdPrm owner) { flags |= 1; fromOwner = UInt8(owner); }

        StdPrm GetFromData() const { if ((flags&1)==0) return 11; return fromData;
} // "11" is kObjectID field of objects
        void SetFromData(StdPrm data) { flags |= 1; fromData = data; }
};

@       PushAction

class PushActionParam {
public:
        UInt8 interactionIndex;
        UInt8 dataForInteractingObject; // could reference stack params or stack
locals.
        UInt8 priority;
        UInt8 flags;
        UInt8 stackLocalForIconObject;

        bool GetUseSpecialIcon() const {return (flags&1)?true:false;}
        void SetUseSpecialIcon(bool use) {flags&=~1;if (use) flags|=1;}

        bool GetUseLocal() const {return (flags&2)?true:false;}
```

```
        void SetUseLocal(bool use) {flags&=~2;if (use) flags|=2;}

        bool GetContinue() const {return (flags&4)?true:false;}
        void SetContinue(bool cont) {flags&=~4;if (cont) flags|=4;}

        bool GetCarryNameOver() const {return (flags&8)!=0;}
        void SetCarryNameOver(bool carry) {flags&=~8;if (carry) flags|=8;}

        // priority values
        enum {
                kInherited = 0,
                kMaximum = 1,
                kAutonomous,
                kUserDriven
        };
};

@       FindFunctionalObject

// functions used for both kFindFunctionalObject and kCallFunctionalTree
enum {
        kFunctionPrepare,
        kFunctionCook,
        kFunctionSurface,
        kFunctionDispose,
        kFunctionFood,
        kFunctionPickupFromSlot,
        kFunctionWashDish,
        kFunctionEatingSurface,
        kFunctionSit,
        kFunctionStand,
        kFunctionServingSurface,
        kFunctionClean,
        kFunctionGarden,
        kFunctionWashHands,
        kFunctionRepair
};

class FindFunctionalObjectParam {
public:
        StdPrm whichFunction;
};

@       TreeBreak

class TreeBreakParam {
public:
        StdPrm  checkData,
                        checkOwner;             // value to check for non-zero
};

@       FindGoodLocation

class FindGoodLocationParam {
public:
        UInt8 specialCondition;
        UInt8 relObjLocal;
        UInt8 flags;

        enum {
                kSC_None,
                kSC_OutOfWorld,
                kSC_SmokeCloudForFight,
                kSC_AlongObjectVector,
                kSC_LateralToObjectVector
```

```
        };

        bool GetStartAtRelObj() const {return (flags&1)!=0;}
        void SetStartAtRelObj(bool start) {flags &= ~1; if (start) flags |= 1;}

        bool GetPreferEmptyTiles() const {return (flags&2)==0;} // watch the not !
        void SetPreferEmptyTiles(bool prefer) {flags &= ~2; if (!prefer) flags |=
2;}

        bool GetEditableTilesOnly() const {return (flags&4)!=0;}
        void SetEditableTilesOnly(bool flag) {flags &= ~4; if (flag) flags |= 4;}
};

@       IdleForInput

class IdleForInputParam {
public:
        StdPrm          decParam;       // which local to decrement
        StdPrm          interruptable;  // can we push the stack while in this
primitive?
};

@       KillObject

class KillObjectParam {
public:
        StdPrm who;
        UInt8 flags;

        bool GetReturnImmediately() const {return (flags&1)!=0;}
        void SetReturnImmediately(bool flag) {flags&=~1; if (flag) flags|=1;}

        bool GetCleanupAll() const {return (flags&2)==0;}
        void SetCleanupAll(bool flag) {flags&=~2; if (!flag) flags|=2;}
};

@       MakeNewCharacter

class MakeNewCharacterParam {
public:
        UInt8 localForSkinColor;
        UInt8 localForAge;
        UInt8 localForGender;
};

@       CallFunctionalTree

class CallFunctionalTreeParam {
public:
        StdPrm whichFunction;
        StdPrm flags;

        bool GetChangeIcon() const {return (flags&1)?true:false;}
        void SetChangeIcon(bool change) {flags&=~1;if (change) flags|=1;}
};

@       ShowString

class ShowStringParam {
public:
        StdPrm stringsID;
        StdPrm stringIndex;
};

@       LookTowards
```

```
class LookTowardsParam {
public:
        // Possible values for typeOfLook
        enum {
                kHeadToStackObject=0,
                kBodyToCamera,
                kBodyToStackObject,
                kBodyAwayFromStackObject
        };

        StdPrm typeOfLook;
};

@       PlaySound

class PlaySoundParam {
public:
        StdPrm  soundID;
        UInt16  sampleRate; // 8.8 fixed point
        UInt8   flags;
        SInt8   volume;         // +- added to base volume (normalized -100..100)

        void SetSampleRate(float sr) {sampleRate = UInt16(sr*256);}
        float GetSampleRate() const {
                return (sampleRate) ? (float(sampleRate)/256) : (1.0f); //
sampleRate 0 is also default 1.0
        }

        bool GetLooped() const {return (flags&1)?true:false;}
        void SetLooped(bool looped) {flags&=~1; if (looped) flags|=1;}

        bool GetUseStackObjAsSource() const {return (flags&2)?true:false;}
        void SetUseStackObjAsSource(bool b) {flags&=~2; if (b) flags|=2;}

        // Whether or not the sound is affected by house viewer zoom level.
        bool GetZooms() const {return (flags&4)?false:true;}
        void SetZooms(bool b) {flags&=~4; if (!b) flags|=4;}

        bool GetPans() const {return (flags&8)?false:true;}
        void SetPans(bool b) {flags&=~8; if (!b) flags|=8;}

        bool GetAutoVary() const {return (flags&16)?true:false;}
        void SetAutoVary(bool autoVary) {flags&=~16; if (autoVary) flags|=16;}

        bool GetModifyRateBasedOnSimSpeed() const {return (flags&32)?false:true;}
        void SetModifyRateBasedOnSimSpeed(bool modify) {flags&=~32; if (!modify)
flags|=32;}
};

@       Relationship

class RelationshipParam {
public:

        // Values for whoseRelationship field.
        enum {
                kMeToStackObject=0,
                kStackObjectToMe,
                kStackObjectToObjInStackVar0,
                kObjInStackVar0ToStackObject
        };

        SInt8 doSet;
        SInt8 index;
```

```
        SInt8 stackVar;
        SInt8 whoseRelationship;
        SInt8 flags;

        bool GetCreate() const {return (flags&1)?false:true;}
        void SetCreate(bool create) {flags&=~1; if (!create) flags|=1;}

        bool GetUseNeighbors() const {return (flags&2)?true:false;}
        void SetUseNeighbors(bool isneighbor) {flags&=~2; if (isneighbor)
flags|=2;}
};

@       Budget

class BudgetParam {
public:
        // Old owners.
        // If oldAmountOwner is set to kBdg_NormalOwnerData, the amountOwner field
is used.
        enum {
                kBdg_Literal,
                kBdg_StackVar,
                kBdg_StackLocal,
                kBdg_Normal
        };

        UInt8 oldAmountOwner;
        UInt8 amountOwner;
        StdPrm amountData;
        StdPrm flags;
        UInt8 expType; // ExpenseType

        // "just test" is true if the budget should only be tested against the
amoun.
        bool GetJustTest() const {return (flags&1)?true:false;}
        void SetJustTest(bool justTest) {flags &= ~1; if (justTest) flags|=1;}

        bool GetSubtract() const {return (flags&2)?false:true;}
        void SetSubtract(bool sub) {flags&=~2; if (!sub) flags|=2;}

        // Automatically converts.
        UInt8 GetAmountOwner() const {
                switch (oldAmountOwner) {
                case kBdg_Literal: return 7; // kImmediate
                case kBdg_StackVar: return 9; // kStackParams
                case kBdg_StackLocal: return 25; // kStackLocals
                default:
                case kBdg_Normal: return amountOwner;
                }
        }
};

        Relationship2

struct Relationship2Param {

        // Values for whichRelationship field.
        enum {
                kMeToStackObject=0,
                kStackObjectToMe,
                kStackObjectToObjInLocal,
                kObjInLocalToStackObject
        };

        UInt8 whichVar;
```

```
        UInt8 whichRel;
        UInt8 flags2;
        UInt8 localForObject;
        UInt8 owner;
        UInt8 _pad;
        StdPrm data;

        bool GetCreate() const {return (flags2&1)?false:true;}
        void SetCreate(bool create) {flags2&=~1; if (!create) flags2|=1;}

        bool GetUseNeighbors() const {return (flags2&2)?true:false;}
        void SetUseNeighbors(bool isneighbor) {flags2&=~2; if (isneighbor)
flags2|=2;}

        bool GetAssign() const {return (flags2&4)?true:false;}
        void SetAssign(bool doSet) {flags2&=~4; if (doSet) flags2|=4;}
};

@       GotoRelative

enum {
        kOnTopLocation=-2,      // stand on top of it
        kDontCareLocation=-1,   // signifies any side is OK
        kDontCareDirection=-1,  // signifies any facing is OK
        kFacingDirection=-2             // only facing the target object is OK
};

class GotoRelativeParam {
public:
        StdPrm          oldTrapCount;           // how long to meander around
obstacles
        SInt8           relLocation,    // side of object to go to
                                relDirection;   // relative direction
        StdPrm          routeCountUNUSED; // how many steps to take before
reorienting
        UInt8           flags;  // signal to fail route if other trees are waiting

        bool GetAllowFailureTrees() const { return (flags&2)==0;}
        void SetAllowFailureTrees(bool allow) { flags&=~2; if (!allow) flags|=2;}

        bool GetAllowDifferentAlts() const {return (flags&4)!=0;}
        void SetAllowDifferentAlts(bool allow) {flags&=~4; if (allow) flags|=4;}
};

@       CallNamedTree

class CallNamedTreeParam {
public:
        StdPrm stringsID;
        StdPrm flags;
        UInt8 stringIndex;
        UInt8 how;

        // values for how field
        enum {
                kRunStackObjectsTreeInMe,
                kRunStackObjectsTreeInStackObj,
                kCallStackObjectsTreeInMe
        };

        bool GetNameGlobal() const {return (flags&1)?true:false;}
        void SetNameGlobal(bool global) {flags&=~1; if (global) flags|=1;}

};
```

```
@       SetMotiveDelta

class SetMotiveDeltaParam {
public:
        UInt8 deltaOwner;
        UInt8 maxOwner;
        UInt8 motiveNum;
        UInt8 flags;
        SInt16 deltaData;
        SInt16 maxData;

        bool GetClearAll() const {return (flags & 1)!=0;}
        void SetClearAll(bool clear) {flags &= ~1; if (clear) {flags |= 1;}}
};

@       GosubFoundAction

class GosubFoundActionParam {
public:
};

@       SetToNext

class SetToNextParam {
public:
        // Which item to get the next of.
        enum {
                kAny,
                kPerson,
                kNonPerson,
                kMultiTile,
                kType,
                kNeighbor,
                kCategoryEqualStackVar0,
                kNeighborOfType,
                kObjectOnTile,
                kAdjacentToObjectInLocal,
                kCareer,
                kHouse
        };

        SInt32 guid; // For use with kType.
        UInt8 flags;
        UInt8 targOwner;
        UInt8 local;
        UInt8 targData;

        Int GetSearchType() const {return flags&0x7f;}
        void SetSearchType(Int srchType) {flags=((srchType&0x7f)|(flags&0x80));}

        Int GetTargetOwner() const
                {
                        if (flags&0x80)
                                return targOwner;
                        return 10;
                }

        Int GetTargetData() const
                {
                        if (flags&0x80)
                                return targData;
                        return 0;
                }

        void SetTarget(Int owner, Int data)
```

```
                {
                        flags |= 0x80;
                        targOwner = owner;
                        targData = data;
                }
};

@       TestObjectType

class TestObjectTypeParam {
public:
        SInt32 guid;
        SInt16 idData;
        UInt8 idOwner;
};

@       Find5WorstMotives

class Find5WorstMotivesParam {
public:
        UInt16 unused0;
        UInt16 unused1;
        UInt16 whoToSearch; // 0 - me, 1 - stack obj. 2 - target obj.
        UInt16 typeOfSearch; // 0 - both, 1 - physical only, 2 - mental only
};

@       UIEffect

class UIEffectParam {
public:
        // Only type 0 for now - flashes a button.
        UInt8 type;

        // Specifies additional data.
        // "which" = which button.
        UInt8 whichOwner;
        SInt16 whichData;
        UInt8 flags;

        bool GetTurnOn() const {return (flags&1)!=0;}
        void SetTurnOn(bool on) {flags&=~1; if (on) flags|=1;}
};

@       UserEvent

class UserEventParam
{
public:
        UInt16 timeout; // in seconds.
        UInt8 size;
        UInt8 zoom;
        UInt8 flags;
        UInt8 strIndex;

        bool GetTurnOn() const {return (flags&1)!=0;}
        void SetTurnOn(bool turnOn) {flags &= ~1; if (turnOn) flags|=1;}

        bool GetLiveAction() const {return (flags&2)!=0;}
        void SetLiveAction(bool live) {flags &= ~2; if (live) flags|=2;}

        bool GetShowIfObjectVisible() const {return (flags&4)!=0;}
        void SetShowIfObjectVisible(bool show) {flags &= ~4; if (show) flags|=4;}

        bool GetJustCenter() const {return (flags&8)!=0;}
        void SetJustCenter(bool flag) {flags &= ~8; if (flag) flags|=8;}
```

```
        bool GetSnapshot() const {return (flags&16)!=0;}
        void SetSnapshot(bool flag) {flags &= ~16; if (flag) flags|=16;}

        bool GetTimoutUsesLocal() const {return (flags&32)!=0;}
        void SetTimoutUsesLocal(bool flag) {flags &= ~32; if (flag) flags|=32;}

        bool GetSlowdown() const {return (flags&64)==0;}
        void SetSlowdown(bool flag) {flags &= ~64; if (!flag) flags|=64;}
};

@       Dialog

class DialogParam {
public:
        // Dialog types.
        enum {
                kMessage,
                kChoice,
                kTriChoice,
                kTextEntry,
                kTutorial
        };

        // Icon types
        enum {
                kAutomatic,
                kNone,
                kNeighbor,
                kPrivateIndexed,
                kGlobalNamed
        };

        // Behaviors
        enum {
                kEngageAndPause,
                kReturnAndPause,
                kEngageAndContinue,
                kReturnAndContinue
        };

        enum {
                kIndexedIconBaseID=5000
        };

        // Strings use a 1-based index. 0 means none.

        UInt8   cancelStr;
        union {
                UInt8   iconIndex;
                UInt8   iconNameStr; // id is specified in a string.
        };
        UInt8   messageStr;
        UInt8   yesStr;
        UInt8   noStr;
        UInt8   type;
        UInt8   titleStr;
        UInt8   flags;

        int GetResultTemp() const {return (flags&0x70)>>4;}
        void SetResultTemp(int tmp) {flags&=~0x70; flags|=((tmp<<4)&0x70);}

        int GetIconType() const {return (flags>>1)&0x07;}
        void SetIconType(int type) {flags &= ~0x0e; flags |= (type&0x07)<<1;}
```

```
        int GetBehavior() const {return ((flags&0x80)?2:0) + ((flags&0x01)?1:0);}
        void SetBehavior(int beh) {flags&=~0x81; if (beh&2) flags|=0x80; if
(beh&1) flags|=0x01;}

        bool GetReturnImmediately() const {return (flags&0x01)!=0;}
        bool GetPauseSimulation() const {return (flags&0x80)==0;}
};

@       TestInteractingWith

class TestInteractingWithParam {
public:
};

@       SetBalloon

class SetBalloonParam {
public:
        StdPrm flags2;
        StdPrm indexAndGroup;
        StdPrm duration;
        StdPrm flagsAndType;

        int GetIndex() const {return indexAndGroup&0x00ff;}
        void SetIndex(int index) {indexAndGroup&=0xff00; index&=0x00ff;
indexAndGroup|=index;}

        int GetGroup() const {return (indexAndGroup&0xff00)>>8;}
        void SetGroup(int group) {indexAndGroup&=0x00ff; group<<=8;
indexAndGroup|=group;}

        int GetType() const {return flagsAndType&0x00ff;}
        void SetType(int newType) {flagsAndType&=~0x00ff; flagsAndType|=newType;}

        bool GetShowWhenInactive() const {return (flagsAndType&0x0100)?
true:false;}
        void SetShowWhenInactive(bool show) {flagsAndType&=~0x0100; if (show)
{flagsAndType|=0x0100;}}

        bool GetShowNotSign() const {return (flagsAndType&0x0200)?true:false;}
        void SetShowNotSign(bool show) {flagsAndType&=~0x0200; if (show)
{flagsAndType|=0x0200;}}

        bool GetReverse() const {return (flagsAndType&0x400)?true:false;}
        void SetReverse(bool rev) {flagsAndType&=~0x0400; if (rev)
{flagsAndType|=0x0400;}}

        bool GetDurationIsInLoops() const {return (flagsAndType&0x800)?
true:false;}
        void SetDurationIsInLoops(bool rev) {flagsAndType&=~0x0800; if (rev)
{flagsAndType|=0x0800;}}

        bool GetOffsetByTemp0() const {return (flagsAndType&0x1000)?true:false;}
        void SetOffsetByTemp0(bool offset) {flagsAndType&=~0x1000; if (offset)
{flagsAndType|=0x1000;}}

        bool GetOverStackObject() const {return (flags2&1)!=0;}
        void SetOverStackObject(bool over) {flags2 &= ~1; if (over) flags2|=1;}

        StdPrm GetLocalNum() const {return (flags2&0x07e0)>>1;}
        void SetLocalNum(StdPrm localNum) {flags2 &= ~0x07e0; flags2|=
(localNum<<1)&0x07e0;}
};

@       CreateObject
```

```
class CreateObjectParam {
public:
        SInt32  guid;
        SInt8   where;
        SInt8   flags;
        SInt8   local;

        bool GetNoDuplicate() const {return (flags&1)?true:false;}
        void SetNoDuplicate(bool noDup) {flags&=~1; if (noDup) flags|=1;}

        bool GetPassMyIDAndStackObjID() const {return (flags&2)?true:false;}
        void SetPassMyIDAndStackObjID(bool set) {flags&=~2; if (set) flags|=2;}

        bool GetCreateNeighborInStackObj() const {return (flags&4)?true:false;}
        void SetCreateNeighborInStackObj(bool neigh) {flags&=~4; if (neigh)
flags|=4;}

        bool GetEmptyTilesOnly() const {return (flags&8)?true:false;}
        void SetEmptyTilesOnly(bool only) {flags&=~8; if (only) flags|=8;}

        bool GetPassTemp0() const {return (flags&16)?true:false;}
        void SetPassTemp0(bool only) {flags&=~16; if (only) flags|=16;}

        bool GetUseDirectionOfStackObj() const {return (flags&32)?true:false;}
        void SetUseDirectionOfStackObj(bool face) {flags&=~32; if (face)
flags|=32;}

        enum {
                // where values
                kInFrontOfMe=0,
                kOnTopOfMe,
                kInMyHand,
                kInFrontOfStackObj,
                kInStackObjSlot0,
                kBelowMe,
                kOutOfWorld,
                kBelowObjectInStackVar0,
                kBelowObjectInLocal,
                kNextToMeInDirectionOfLocal
        };
};

@       DropOnto

class DropOntoParam {
public:
        StdPrm srcInStackVar;
        StdPrm src;
        StdPrm destInStackVar;
        StdPrm dest;
};

@       AnimateNew

class AnimateNewParam {
public:
        StdPrm animID;
        UInt8 localNumForEvent;
        UInt8 _pad;
        SInt8 source;
        SInt8 flags;
        SInt8 expectedEventCount;

        bool GetBackwards() const {return (flags&2)?true:false;}
```

```
        void SetBackwards(bool flag) {flags&=~2; if (flag) flags|=2;}

        bool GetUsesStackVar() const {return (flags&4)?true:false;}
        void SetUsesStackVar(bool flag) {flags&=~4; if (flag) flags|=4;}

        bool GetInterruptible() const {return (flags&8)?true:false;}
        void SetInterruptible(bool flag) {flags&=~8; if (flag) flags|=8;}

        bool GetHurryable() const {return (flags&64)?true:false;}
        void SetHurryable(bool flag) {flags&=~64; if (flag) flags|=64;}

        bool GetUseLocalForEvents() const {return (flags&32)?true:false;}
        void SetUseLocalForEvents(bool flag) {flags&=~32; if (flag) flags|=32;}

        Int GetBehavior() const {
                Int beh = 0;
                if (flags&1) {
                        beh|=1;
                }
                if (flags&16) {
                        beh|=2;
                }

                return beh;
        }
        void SetBehavior(Int beh) {
                flags &= ~(1|16);

                if (beh&1) {
                        flags |= 1;
                }
                if (beh&2) {
                        flags |= 16;
                }
        }

        // Possible values for source
        enum {
                kObject,
                kGlobal,
                kPerson,
                kMisc
        };

        // Possible values for behavior.
        enum {
                kNormal,
                kBackground,
                kCarry,
                kNormal_CarryOverride,
        };
};

@       GotoRoutingSlot

class GotoRoutingSlotParam {
public:
        StdPrm paramValue;
        StdPrm paramType;
        UInt8 flags;

        bool GetAllowFailureTrees() const {return (flags&1)==0;}
        void SetAllowFailureTrees(bool allow) {flags&=~1; if (!allow) flags|=1;}

        // values for paramType;
```

```
        enum {
                kStackVar,
                kLiteralSlot,
                kGlobalSlot
        };
};

@       Snap

enum {
        kSnapToRoutingSlotInStackVar,
        kSnapOnTop,
        kSnapInFront,
        kSnapToLiteralSlotNum,
        kSnapToGlobalSlotNum
};

class SnapParam {
public:
        StdPrm stackVarOfRoutingSlot;
        StdPrm howToSnap;
        StdPrm flags;

        bool GetOriginOnly() const {return (flags&1)!=0;}
        void SetOriginOnly(bool only) {flags &= ~1; if (only) flags|=1;}

        bool GetAskPersonToMove() const {return (flags&2)!=0;}
        void SetAskPersonToMove(bool ask) {flags &= ~2; if (ask) flags|=2;}

        bool GetUseFootprint() const {return (flags&4)!=0;}
        void SetUseFootprint(bool flag) {flags &= ~4; if (flag) flags|=4;}
};

@       Reach

enum {
        kReachToStackObject=0,
        kReachToSlotOfStackObject,
        kReachToMouth,
};

class ReachParam {
public:
        StdPrm whereToReach;
        StdPrm shouldGrabOrDrop;
        StdPrm stackVarOfSlotNumber;
};

@       KillSounds

class KillSoundsParam {
public:
        StdPrm useStackObject;
};

@       NotifyStackObject

class NotifyStackObjectParam {
public:
};

@       MakeActionString

class MakeActionStringParam {
public:
```

```
            StdPrm stringsID;
            StdPrm flags;
            UInt8 stringIndex;
    };

    @
```