

MASSIVE-3 Core API excerpts, 28/6/99

FILE AGENT.H	2
Type AgentMainloopCBLevel	2
Type AgentMainloopCBLevelValues	2
Type AgentMainloopCB	2
Class Agent	3
FILE ENVIRONMENT.H	5
Class EnvironmentItemChildIterator	6
Class EnvironmentItem	7
Class EnvironmentItemIterator	8
Class EnvironmentAPI	9
Class Environment	11
Class RequestForOwnership	14
Class ControlledItemAPI	15
FILE ENVCALLBACK.H	16
Type EnvCallbackFn	16
Class EnvCallback	17

File Agent.h

```
//=====
// Agent.h
//=====
```

Type AgentMainloopCBLevel

```
// Agent mainloop callback levels
typedef int AgentMainloopCBLevel;
```

Type AgentMainloopCBLevelValues

```
// Standard mainloop callback levels (= priorities/order in main loop)
typedef enum
{
    AGENT_CBLEVEL_FIRST=0,
    AGENT_CBLEVEL_SENDING=3,
    AGENT_CBLEVEL_NETWORK=6,
    AGENT_CBLEVEL_PENDING=9,
    AGENT_CBLEVEL_INTERACTION=12,
    AGENT_CBLEVEL_POST_INTERACTION=15,
    AGENT_CBLEVEL_RENDER=18,
    AGENT_CBLEVEL_LAST=21
} AgentMainloopCBLevelValues;
```

Type AgentMainloopCB

```
// Agent mainloop callback method type
//   cbLevel = main loop level at which callback was called.
//   frameStartTime = machine local time sampled at start of frame.
//   userData = user data passed to addMainloopMethod/Timer.
typedef void (*AgentMainloopCB)(AgentMainloopCBLevel cbLevel,
                                Time &frameStartTime,
                                void *userData);
```

Class Agent

```
// Agent class, encapsulates thread and activity
class HIVEKEEXPORT Agent : public Root {
    //=====
    // Agent class primary API
public:

    //-----
    // Object life-cycle

    // Standard constructor: allocates new ID for Agent, and registers
    // Agent with specified name. NetworkEventHandler must already
    // exist.
    Agent(NetworkEventHandler *eh, crg_string_t name = NULL);

    // Agent Destructor - not currently guaranteed to be garbage free
    // (it is expected that an Agent will normally live for as long as
    // its hosting process).
    ~Agent();

    //-----
    // Object getters/setters

    // returns the id of this Agent
    void getId(AgentID& id);

    // given an EnvironmentID or Host-IP/Name string, return a pointer to the
    // environment if this Agent has a replica of it.
    Environment* findEnvironment(EnvironmentID& env_id);
    Environment* findEnvironment(crg_string_t name);

    //-----
    // Agent Environment management

    // Join an Environment specified as "host-IP/name".
    // If host-IP is not specified (e.g. "/name" or "name") then defaults
    // to local host.
    // The actual Environment instance is created by EnvironmentFactory
    // (see setEnvironmentFactory).
    // Returns NULL if the attempt fails.
    Environment* joinEnvironment(crg_string_t env_name);

    // join an Environment with specified ID.
    // The actual Environment instance is created by EnvironmentFactory
    // (see setEnvironmentFactory).
    // Returns NULL if the attempt fails.
    Environment* joinEnvironment(EnvironmentID& id);

    // Leave specified Environment
    crg_bool_t leaveEnvironment(Environment* e);

    // Create a new environment with the master as this agent.
    // The actual Environment instance is created by EnvironmentFactory
    // (see setEnvironmentFactory).
    Environment* createEnvironment(EnvironmentID& id,
                                   crg_string_t env_name = NULL,
                                   crg_bool_t total_order = 0,
                                   AudioType audio_type=PEER_TO_PEER_UNICAST);

    // Change environment factory to be used by createEnvironment,
    // returns old environment factory.
    // This is used to change the actual Environment (sub)class created
    // by joinEnvironment or createEnvironment. E.g. additional
    // environment functionality (such as audio-support) may be
    // provided by a particular subclass of Environment.
    EnvironmentFactory *setEnvironmentFactory(EnvironmentFactory *fact);
};
```

```

// Add an environment to the list managed by this agent and register it
with
// the local trader. Used when Environments are created in strange ways -
// like from a file.
void addEnvironment(Environment* env);

//-----
// Timer and main loop functionality

// Add default main loop functions - sending, network, pending
void addDefaultMainloopMethods();

// add a mainloop method (called every frame)
void addMainloopMethod(AgentMainloopCBLevel cbLevel,
                      AgentMainloopCB callback,
                      void *user_data);

// Add a timer method (called after delay (repeatFlag=0),
// or every delay (repeatFlag!=0).
void addMainloopMethod(AgentMainloopCBLevel cbLevel,
                      AgentMainloopCB callback,
                      void *user_data,
                      Time &delay, int repeatFlag);

// Remove a mainloop/timer method - arguments must match the
// corresponding call to addMainloopMethod exactly.
void removeMainloopMethod(AgentMainloopCBLevel cbLevel,
                          AgentMainloopCB callback,
                          void *user_data);

// Remove all mainloop methods which match the specified callback
// and/or user data.
void removeMainloopMethods(AgentMainloopCB callback=NULL,
                           void *user_data=NULL);

// Do main loop - may never return. Optionally specify minimum
// interframe time (in seconds and microseconds). This may be
// over-riden to make it shorter (for internal system liveness
// on some platforms). maxDelaySec < 0 requests indefinite
// blocking when there are no network or timer events.
void mainloop(long maxDelaySec=-1, long maxDelayUsec=0);

// quit mainloop - at end of current frame. Typically called
// automatically by standard signal handler.
void quitMainloop();

//=====
// Supplementary API methods

// ...
};

```

File Environment.h

```
//=====
// Environment.h
//=====
```

Class EnvironmentItemChildIterator

```
/*
   class to iterate children of EnvironmentItem
*/
class HIVEKEEXPORT EnvironmentItemChildIterator
{
    //=====
    // EnvironmentItemIterator primary API
public:

    // Default constructor
    inline EnvironmentItemChildIterator()
        : nextElem(0)
    {}

    // Initialise using EnvironmentItem::iterateChildren

    // Get next child. Note - unsafe to call across changes in
    // children.
    EnvironmentItem *next();

    //=====
    // end of EnvironmentItemIterator primary API

    // .....
};
```

Class EnvironmentItem

```
/*
   Class containing information about a single item in the environment.
   The actual item data is stored in another object referenced by this one.
*/

class HIVEKEXPORT EnvironmentItem : public Serializable {
    //=====
    // EnvironmentItem primary API
public:

    //-----
    // getters/setters

    // Iterate children of this item
    EnvironmentItemChildIterator *
    iterateChildren(EnvironmentItemChildIterator* i);

    // Get item data
    ItemData *getItem();

    // Get current owner
    AgentID getOwner();

    // Get responsible agent (important for process bound items)
    AgentID getResponsible();

    // Get lock type
    LockType getLocked();

    // Get sequencer values
    Sequencer getMseq();
    Sequencer getOseq();

    // Get Environment hosting this item
    Environment* getEnvironment();

    //=====
    // end of EnvironmentItem primary API
    // ....
};
```

Class EnvironmentItemIterator

```
/*
   Class used to iterate through the contents of an environment
*/
class HIVEKEEXPORT EnvironmentItemIterator : public HashTableIterator {
    //=====
    // EnvironmentItemIterator primary API
public:

    // Default constructor.
    // Then initialise using Environment::iterateTopLevel or
    // Environment::iterate
    inline EnvironmentItemIterator()
    : HashTableIterator(NULL) {
    }

    // Get next item in iteration.
    // Note - not safe to call across changes to the Environment.
    inline EnvironmentItem* next() {
        HashTableElement* he = HashTableIterator::next();
        return (EnvironmentItem*) ((he != NULL) ? he->data : NULL);
    }

    //=====
    // EnvironmentItemIterator supplementary API

    // ....
};
```

Class EnvironmentAPI

```
/*
   The abstract interface to the environment with which an object can
   interact - used by Environment and also by RequestForOwnership to
   stack up events subject to a transfer of ownership
*/

class HIVEKEXPORT EnvironmentAPI
{
    //=====
    // EnvironmentAPI primary API
public:

    //-----
    // Standard Event generators

    // Generate item ADD Event
    // Note: takes ItemData.
    // Note: used by addNewX utility methods (in Environment)
    Status addItem(ItemData* item, LockType locked=LOCK_HARD);
    Status addItem(ItemData* item, LockType locked, Time &minTime);
    // note: steals constraints list contents
    Status addItem(ItemData* item, LockType locked, Time &minTime,
                   List& constraints);

    // Generate item UPDATE Event
    // Note: takes ItemData.
    // Note: used by updateX utility methods (in Environment)
    Status itemUpdate(ItemData* item);
    // Make UPDATE optional rather than mandatory (which is default)
    Status itemUpdateOptional(ItemData* item);
    Status itemUpdate(ItemData* item, Time &minTime);
    // note: steals constraints list contents
    Status itemUpdate(ItemData* item, Time &minTime, List& constraints,
                      long optional=0);

    // Generate item DELETE Event
    Status itemDelete(ItemID& id);
    Status itemDelete(ItemID& id, Time &minTime);
    // note: steals constraints list contents
    Status itemDelete(ItemID& id, Time &minTime, List& constraints);

    // Do depth-first iteration of item tree, generating DELETE Events
    // for all items.
    Status itemDeleteTree(ItemID& id); // Delete item and children.
    Status itemDeleteTree(ItemID& id, Time &minTime);

    // Generate item SET_OWNER Event
    Status itemSetOwner(ItemID& id, AgentID& new_owner,
                       LockType locked=LOCK_NONE);
    Status itemSetOwner(ItemID& id, AgentID& new_owner,
                       LockType locked, Time &minTime);
    // note: steals constraints list contents
    Status itemSetOwner(ItemID& id, AgentID& new_owner,
                       LockType locked, Time &minTime,
                       List& constraints);

    // Generate item SET_LOCKED Event
    Status itemSetLocked(ItemID& id, LockType locked=LOCK_HARD);

```

```

// Generate NOTIFICATION Event
Status notification(EventNotificationData *message);
Status notification(EventNotificationData *message, Time &minTime);
// note: steals constraints list contents
Status notification(EventNotificationData *message, Time &minTime,
                    List& constraints);

// generate LOG_MESSAGE events

Status logMessage(EventLogMessageData *message);
Status logMessage(EventLogMessageData *message, Time &minTime);
// note: steals constraints list contents
Status logMessage(EventLogMessageData *message, Time &minTime,
                    List& constraints);

// Mark start and end of atomic actions.
// Note: not currently supported.
void startAtomic();
void endAtomic();

//=====
// end of EnvironmentAPI primary API

// ....
};

```

Class Environment

```
class HIVEKEEXPORT Environment : public Serializable, public EnvironmentAPI {
    //=====
    // EnvironmentAPI primary API
public:

    //-----
    // Object life-cycle

    // Environment objects are normally created by an EnvironmentFactory
    // at the request of an Agent: See
    //   Agent::joinEnvironment
    //   Agent::createEnvironment
    //   Agent::leaveEnvironment

    //-----
    // getters and setters

    // Get Environment's ID
    inline EnvironmentID getID() {return id;}

    // Get ID of Agent responsible for this local Environment replica
    AgentID getOwnerID();

    // Get ID of master Agent responsible for this Environment.
    AgentID getMasterID();

    // Get reference to Agent responsible for this local Environment
    // replica. Normally this will be the only local Agent.
    Agent *getOwner();

    // Get Environment name
    inline crg_string_t getName() {return name;}

    //-----
    // Item ID management

    // Return a new item ID. For a non-master Environment this may
    // involve an RPC request.
    Status getNewID(ItemID& new_id);

    // Return a block of new item IDs. item_id++ for each one. For a
    // non-master Environment this may involve an RPC request.
    Status getNewIDs(crg_int32_t num_ids, ItemID& first_id);

    //-----
    // Item manipulation convenience functions.
    // This wrap up calls to EnvironmentAPI methods.

    // utility function for specialised itemAdd operations
    ItemID addNewEntity(ItemID *parent=NULL, Matrix3D *transform=NULL,
                       Extent3D *extent=NULL,
                       Trajectory3D *trajectory=NULL,
                       LockType lock=LOCK_HARD,
                       crg_int32_t flags=ITEM_PROCESS_BOUND);
    ItemID addNewAttribute(ItemID *parent, char *name, char *value,
                          LockType lock=LOCK_HARD,
                          crg_int32_t flags=ITEM_PROCESS_BOUND);
    ItemID addNewFilter(ItemID *parent, char *name, char *value,
                       LockType lock=LOCK_HARD,
                       crg_int32_t flags=ITEM_PROCESS_BOUND);
    ItemID addNewGeometry(ItemID *parent, char *url,
                          LockType lock=LOCK_HARD,
                          crg_int32_t flags=ITEM_PROCESS_BOUND);
    ItemID addNewSwitch(ItemID *parent, int value=0,
                       LockType lock=LOCK_HARD,
```

```

        crg_int32_t flags=ITEM_PROCESS_BOUND);
ItemID addNewLink(ItemID *parent, ItemID *target,
        LockType lock=LOCK_HARD,
        crg_int32_t flags=ITEM_PROCESS_BOUND);
ItemID addNewBoundary(ItemID *parent, char *targetName,
        char *abstractionName=NULL,
        Polygon3D *polygon=NULL,
        Matrix3D *transform=NULL,
        float awarenessTransform=1.0,
        ItemID *reciprocal=NULL,
        LockType lock=LOCK_HARD,
        crg_int32_t flags=ITEM_PROCESS_BOUND);

// utility functions for specialised itemUpdate operations
Status updateEntity(ItemID &id, Matrix3D *newTransform=NULL,
        Extent3D *newExtent=NULL,
        Trajectory3D *newTrajectory=NULL,
        crg_int32_t setFlags=0L,
        crg_int32_t clearFlags=0L,
        int optionalFlag=0);
Status updateEntityOptional(ItemID &id, Matrix3D *newTransform=NULL,
        Extent3D *newExtent=NULL,
        Trajectory3D *newTrajectory=NULL,
        crg_int32_t setFlags=0L,
        crg_int32_t clearFlags=0L);
Status updateAttribute(ItemID &id, char *newName=NULL,
        char *newValue=NULL,
        crg_int32_t setFlags=0L,
        crg_int32_t clearFlags=0L,
        int optionalFlag=0);
Status updateFilter(ItemID &id, char *newName=NULL,
        char *newValue=NULL,
        crg_int32_t setFlags=0L,
        crg_int32_t clearFlags=0L,
        int optionalFlag=0);
Status updateGeometry(ItemID &id, char *newUrl,
        crg_int32_t setFlags=0L,
        crg_int32_t clearFlags=0L,
        int optionalFlag=0);
Status updateSwitch(ItemID &id, int newValue=0,
        crg_int32_t setFlags=0L,
        crg_int32_t clearFlags=0L,
        int optionalFlag=0);
Status updateLink(ItemID &id, ItemID *newTarget,
        crg_int32_t setFlags=0L,
        crg_int32_t clearFlags=0L,
        int optionalFlag=0);
Status updateBoundary(ItemID &id, char *newTargetName=NULL,
        char *newabstractionName=NULL,
        Polygon3D *newPolygon=NULL,
        Matrix3D *newTransform=NULL,
        float newAwarenessTransform=(-1.0),
        ItemID *newReciprocal=NULL,
        crg_int32_t setFlags=0L,
        crg_int32_t clearFlags=0L,
        int optionalFlag=0);

//-----
// Item-related getters

// Get current value of item.
// Returns NULL if item not found.
ItemData* itemGet(ItemID& id);

// Get current EnvironmentItem for item.
// Returns NULL if item not found.
EnvironmentItem* envItemGet(ItemID& id);

```

```

// Get current owner and/or lock type of item.
// Returns STATUS_OK if successful.
Status itemGetOwner(ItemID& id, AgentID& owner, LockType& locked);
Status itemGetOwner(ItemID& id, AgentID& owner);
Status itemGetLocked(ItemID& id, LockType& locked);

// Get transform of specified item within Environment coordinate
// system. Returns 0 if OK (i.e. item and all ancestors found).
int getTransform(ItemID &id, Matrix3D &transform);

//-----
// Item ownership facilities

// Request ownership of item. Request is reified in a
// RequestForOwnership object. Specified callbacks are invoked when
// ownership is transferred or request is rejected.
//   locked = desired lock type on successful ownership transfer.
RequestForOwnership* itemRequestOwnership(ItemID id, LockType locked,
RequestForOwnershipCB
success_cb,
RequestForOwnershipCB
failure_cb,
void *userData);

// Construct a ControlledItemAPI for a (normally CONTROL locked) item.
// This object will implement itemUpdate events using an implicitly
// created UpdateRequest item. This is deleted when the
// ControlledItemAPI is deleted.
ControlledItemAPI* getControlledItemAPI(ItemID id);

//-----
// Environment time management

// Get current globally synchronised time for this Environment.
// Normally different from host local time for non-master
// Environments.
Time getCurrentTime();

//=====
// end of EnvironmentAPI primary API

// ....
};

```

Class RequestForOwnership

```
/*
  Pending request for ownership class
  */
class HIVEKEEXPORT RequestForOwnership : public Root, public EnvironmentAPI
{
public:
  //=====
  // RequestForOwnership primary API

  //-----
  // Object lifecycle

  // RequestForOwnership is only created by
  // Environment::itemRequestOwnership

  // Standard destructor. Event if deleted before any callback,
  // ownership transfer may still occur - events are likely to be in
  // transit.
  ~RequestForOwnership();

  //-----
  // Item ownership facilities

  // RequestForOwnership will fail on a LOCK_CONTROL item
  // -> you might want to do this instead.
  // Passes on pending event(s) to ControlledItemAPI.
  ControlledItemAPI* convertToControlledItemAPI();

  //=====
  // end of RequestForOwnership primary API

  // ....
};
```

Class ControlledItemAPI

```
/*
  ControlledItemAPI - updates only - on a LOCK_CONTROL item
  */
class HIVEKEEXPORT ControlledItemAPI : public EnvironmentAPI
{
public:
  //=====
  // ControlledItemAPI primary API

  //-----
  // Object lifecycle

  // ControlledItemAPI is only created by
  // Environment::getControlledItemAPI or
  // RequestForOwnership::convertToControlledItemAPI

  // Standard destructor. Will delete implicitly created UpdateRequest
  // item if present.
  ~ControlledItemAPI();

  //=====
  // end of ControlledItemAPI primary API

  // .....
};
```

File EnvCallback.h

```
//=====
// EnvCallback.h
//=====
```

Type EnvCallbackFn

```
// Callback handler type signature.
// env = Environment with which callback was registered.
// callback = reference to EnvCallback object itself.
// type = type of event causing callback.
// envItem = reference to event-related item (if any).
// event = reference to event causing the callback.
// userData = userData registered when EnvCallback created.
typedef void (*EnvCallbackFn) ( Environment *env,
                               EnvCallback *callback,
                               EventType type,
                               EnvironmentItem *envItem,
                               Event *event,
                               void *userData );

// generic environment callback - create object to establish,
// delete object to remove.

// level mask, relative to filter ID
// all levels in sub-tree
#define ENV_CALLBACK_ALL_LEVELS (~0)
// filter object only
#define ENV_CALLBACK_OBJECT 0x0001
// children of object only
#define ENV_CALLBACK_CHILDREN 0x0002
```

Class EnvCallback

```
class EnvCallback : public Root {
    //=====
    // EnvCallback class primary API
public:

    //-----
    // Object life-cycle

    // Standard constructor.
    // Note, not all combinations of mask values may yield valid callbacks.
    // Takes responsibility for freeing notificationFilter (if specified).
    // filterID, if specified, is highest item in hierarchy of
    // event-related items considered. NULL = top level.
    // levelMask determines event-related items considered, relative
    // to filterID.
    // beforeDeliveryFlag =1 causes callback to occur before event is
    // delivered (except for DELETE).
    //
    // Note: event type EVENT_ADD_PRESENT will generate immediate callbacks
    // for existing items which match the filter specification.
    EnvCallback(Environment *env, EnvCallbackFn callbackFn, void *userData,
                int eventTypeMask=EVENT_ANY, ItemID *filterID=NULL,
                unsigned long levelMask=ENV_CALLBACK_ALL_LEVELS,
                int itemTypeMask=ITEM_ANY,
                int beforeDeliveryFlag=0,
                EventNotificationData *notificationFilter=NULL);

    // Standard destructor - unregisters callback.
    //
    // Note: event type EVENT_DELETE_PRESENT will generate immediate callbacks
    // for still-existing items which match the filter specification.
    virtual ~EnvCallback();

    // Indicate the callback should be deleted automaticcally when
    // associated Environment or filter item is deleted.
    void freeOnEnvDeleted();

    //-----
    // Object getters/setters

    // get Environment with which EnvCallback is registered
    Environment *getEnvironment();

    // get userData as passed to constructor
    void *getUserData();

    //=====
    // End of primary API

    // ....
};

//=====
```