Scheduling API

This is a way to separate scheduler, from GBE, based on our discussion last year.

In all cases where people are referenced, it is the responsibility of the caller to perform any notifications of users due to schedule change.

Thought – a lot happens on the scheduled.Event – for the Scheduled Event operations that happen on an existing event, should these be functions performed on the model object? Or should it be an IDD where the functions are above the Event level?

# The API

General design:

* Errors – I chose to return blocking errors as a code, rather than Exception. I’d be OK with changing that. My general assumption was that it’s capable to imply a partial success more easily with a code.
* Warnings – I also used a response value for cases of warnings so that the scheduler can control over book and other resource constraint logic. I am less willing to change this to an exception model – exceptions should only be used for interrupts.
* Style – I chose a functional style (get\_, create\_, update\_, add\_, remove\_) – equally valid would be and object oriented design. I don’t have a strong sense of what is most pythonic here.
* Similarly – is it better to use dicts to totally isolate objects, or to pass objects with the properties listed here as return values. Whatever we do, I’d like to be consistent.

## Scheduled Event Operations

Functions that center around the scheduled event object/concept. In the current model, this is scheduler.Event. I’m calling this “scheduled event” to distinguish it from the gbe.Event.

Name change = occurrence = scheduled event

### Search Operations

#### DONE - get\_occurrences

Get all the events that match the given label. Good for the sets needed for both calendars and lists. Having more than one input implies an additive filter. For example, a label + a day = only the events with that label, on that day.

**Inputs** (at least 1 or any combination):

* **labels** – a string for the label identifier.
* **day -** date, optional – to limit the results to only start times within that day. Can be a timestamp, the hh:mm:ss is disregarded.
* **parent\_event\_id** – the unique identifier of the occurrence that is the parent of these children
* **foreign\_event\_id** – the gbe.Event that this occurrence represents.
* **max\_volunteer –** the minimum size of the max\_volunteer value. If None then it doesn’t matter. This is functionally equivalent to “0” which is gte 0 and since the value is always a positive integer… they mean the same.

**Output**:

* a structure with:
  + **occurrences** (required if success) - List of related events. Includes the following sub-elements.
    - **occurrence\_id** – so this can be referenced specifically later
    - **start\_time** – the time, as a timestamp, when the event will start
    - **foreign\_event\_id -** the event whose description is being used, currently this is a gbe.Event.
    - **labels –** list of all descriptor strings
    - **locations** – a list of the places consumed by this event. Must be a list of 0 or more Locations.
    - **people** – a list of the people involved in the event and the relevant details for their participation. See Individual Functions and Group Functions for information on how the “people” structures behave.
  + **code** (required) - The code can be looked up using the UserMessages model or None for success.

#### DONE - get\_occurrence

Gets the info for 1 occurrence, based on the id.

**Inputs** (use just 1):

* **occurrence\_id** – the unique identifier of the occurrence.

**Output**:

* a structure with:
  + **occurrence** (required if success) - List of related events. Includes the following sub-elements.
    - **start\_time** – the time, as a timestamp, when the event will start
    - **foreign\_event\_id -** the event whose description is being used, currently this is a gbe.Event.
    - **locations** – a list of the places consumed by this event. Must be a list of 0 or more Locations.
    - **people** – a list of the people involved in the event and the relevant details for their participation. See Individual Functions and Group Functions for information on how the “people” structures behave.
  + **code** (required if failure) - The code can be looked up using the UserMessages model or None for success.

#### DONE - get\_schedule

Gets the schedule for an entity. Only one entity should be provided, but the different arguments give the options for this function. If python allowed overloading, that’s what I’d do here.

When called, this creates the complete list of bookings for the given target.

**Inputs** (use just 1):

* EITHER NOT BOTH:
  + **user**– the individual. Gets a schedule of all events the person is booked into, either as an individual, or as part of a group.
  + **location –** gets the room schedule.
* **group\_id** – id for a group. Gets only the group’s schedule.
* **start\_time, end\_time** – bookings for events that are after the start time and before the end time. Note – either one may be omitted. Uses python datetime
* **roles** – only if the target is a user. A ist of roles that the user may be allocated to.
* **foreign\_event\_id –** the gbe.Event id. Gets all bookings for that event.

**Optional Input:**

* **labels –** list of labels to limit the search to only those labels.

**Output**:

* a structure with:
  + **schedule\_item** (required if success) – same as above, but with the following variations:
    - **user/group\_id –** either/or – not both.
    - **role (may be empty)–** the role that the person/group is doing for the given event.
    - **label (may be empty) –** the label attached to these commitments.
    - **occurrence (same as above)** – note this includes “people” so it can end up iterative, the people w/in an occurrence are all the people in the occurrence, so you can be quite the stalker here.
  + **code** (required if failure) - The code can be looked up using the UserMessages model or None for success.

### Modification of Events

Functions that modify/create a scheduled event. There’s more options that could be built here – notably, get\_scheduled\_event (based on event id) to get more details, and get\_parent and get\_child.

#### DONE - create\_occurrence

The operation that makes a brand new event at the place and time given. It will check for conflicts and send alerts for people-related conflicts at present.

**Inputs:**

* Required:
  + **foreign\_event\_id** – the event whose description is being used, currently this is a gbe.Event.
  + **start\_time** – the time, as a timestamp, when the event will start
* Optional:
  + **max\_volunteer** – integer describing how many volunteers, defaults to 0
  + **people** – a list of the people involved in the event and the relevant details for their participation. See Individual Functions and Group Functions for information on how the “people” structures behave.
  + **locations** – a list of the places consumed by this event. Must be a list of 0 or more Locations.
  + **parent\_event** – the event (a gbe.Event) that this should be a child to – for example, the Show if this is a tech slot or rehearsal.
  + **labels** (list)–a way to provide a unique tag to aid in searching. This can be used to label the event for what calendar to use, and what list to put it on. Recommendation – put both conference\_slug and calendar type on the label.

**Outputs:**

* A Dictionary:
  + **occurrence** (required if successful) - The scheduler.Event that was created as a result of this request.
  + **warnings** - any warnings. These imply that the creation was successful, but the following alerts should be considered. Each warning can include:
    - **code** (required) - The code can be looked up using the UserMessages model.
    - **user** (User) – only in the cases of a schedule conflict, the user having the conflict.
    - **occurence** (scheduler.Events) – in cases of conflicts, the list of event conflicts.
  + **errors** (required, if failure) – if no event could be provided, this is required.
    - **code –** that can be translated into a UserMessage
    - **details –** a list of strings for targets of the failure

#### DONE - update\_occurrence

Takes an existing scheduled event and updates it. If an optional value is not provided, the existing value remains. To clear a value, another operator could be used, or send ‘None’ as the value.

**Inputs:**

* Required:
  + **occurrence (scheduler.Event)** – the event which is being changed.
* Optional:
  + **start\_time** – the time, as a timestamp, when the event will start
  + **max\_volunteer** – integer describing how many volunteers, defaults to 0
  + **people** – a list of dicts for the people involved in the event and the relevant details for their participation. This is the same structure as *create\_scheduled\_event*. Sending this structure implies that the existing roles should be cleared, and the new ones inserted. “None” means leave the current settings alone.
  + **locations** – a list of the places consumed by this event. Must be a list of 0 or more Locations. . Sending this structure implies that the existing locations should be cleared, and the new ones inserted. “None” means leave the current settings alone.
  + **parent\_event** – the event (a gbe.Event) that this should be a child to – for example, the Show if this is a tech slot or rehearsal. Changes the parent from any previous parents to this one. Sending “None” implies that any existing parents should be cleared. Defaults to “-1” implying “don’t change what’s there”.

**Outputs:**

* A Dictionary:
  + **occurrence** (required if successful) – the updated scheduler.Event that was altered as a result of this request.
  + **warnings** - any warnings. These imply that the creation was successful, but the following alerts should be considered. Each warning can include:
    - **code** (required) - The code can be looked up using the UserMessages model.
    - **user** (User) – only in the cases of a schedule conflict, the user having the conflict.
    - **conflicts** (scheduler.Events) – in cases of conflicts, the list of event conflicts.
  + **errors** (required, if failure) – if no event could be provided, this is required. A UserMessage code relaying the nature of the error.

#### delete\_occurrence

Deletes the scheduled event from the system. Scheduled Event, bookings for people, groups, locations – are all removed.

**Inputs:**

* Required:
  + **occurrence (scheduler.Event)** – the event which is being changed.

**Outputs:**

* + **warnings** - any warnings. These imply that the deletion was successful, but the following alerts should be considered. Each warning can include:
    - **code** (required) - The code can be looked up using the UserMessages model.
  + **errors** (required, if failure) – delete failed. A UserMessage code relaying the nature of the error.

#### delete\_all occurrences

Same as delete\_scheduled\_event, but for the set of events referencing this foreign\_event\_id

**Inputs:**

* Required:
  + **foreign\_event\_id** – the event that all of the scheduled events reference.

**Outputs:**

* + **warnings** - any warnings. These imply that the deletion was successful, but the following alerts should be considered. Each warning can include:
    - **code** (required) - The code can be looked up using the UserMessages model.
  + **errors** (required, if failure) – delete failed. A UserMessage code relaying the nature of the error.

#### check\_conflicts

Don’t assign the item to the event, just check for conflicts.

* Required:
  + **occurrence (scheduler.Event)** – the event which is being changed.
  + **One of the following**:
    - **person –** checks that this person has no scheduling conflicts as either an individual or as a member of any group.
    - **group –** check that every individual within the group does not have a conflict.
    - **act** – check that the person/group represented by the act do not have a conflict.
    - **location** – check that the location does not have any conflicts.

## Human Operations

### Common Functions

These functions work whether or not the object is a person or a group.

#### DONE - get\_bookings

#### DONE (within event) - people

Get all the people (both individuals and groups) affiliated with the event.

**Inputs:**

* Required:
  + **occurrence\_id (scheduler.Event)** – of the event which is being changed.
* Optional:
  + Role = the role given as part of the assignment

**Outputs:**

* **people** – a list of the people involved in the event and the relevant details for their participation. A given item in the list can be either a group, or a person (see the following sections).
* **code** (required) - The code can be looked up using the UserMessages model or None for success.

Leaving an optional thing out is the way to erase it.

#### DONE - remove\_booking

Remove this assignment to the event. Touches only that booking.

**Inputs:**

* Required:
  + **occurrence (scheduler.Event)** – the event which is being changed.
* Optional (one of the following is required):
  + **booking\_id –** integer for the specific booking that will be removed.

**Outputs:**

* **code** (required) - The code can be looked up using the UserMessages model or None for success.

### Individual Functions

Cases of the booking item being 1:1 with a human being. Solo performers, teachers, staff leads – etc.

The construct for a “person” is:

* **booking\_id** (required after creation, int) – identifier that uniquely identified the assignment of THIS person, to THIS event, in THIS way. It’s the resource allocation id.
* **User** (required) – the private id that is a unique identifier of the HUMAN being identified. Used for checking conflicts.
* **public\_id** (optional, int) – the public id for something that can be used to publicize this scheduled event. It’s the responsibility of the caller to be sure that this id has an appropriate relationship with the User. In GBE this would be the persona id. If this is not provided, the person’s role and name won’t be listed.
* **role** (optional, string) – a descriptor for this person’s relationship to this event.
* **label** (optional, string) – a string describing the relationship

#### DONE (within Event) - add\_person,

#### DONE in IDD - set\_booking

Assign people to the event. Assigning the same person twice is allowed, but will cause a conflict warning. Does not change any existing assignments.

**Inputs:**

* Required:
  + **scheduler.Event** – the event which is being changed.
  + **person** – a single item for the people involved in the event and the relevant details for their participation.
    - **No booking id needs to be provided**.

**Outputs:**

* **code** (required) - The code can be looked up using the UserMessages model or None for success.
* **booking\_id -** only on success. The unique integer referencing this assignment of person to event.
* **conflicts** (scheduler.Events) – in cases of conflicts, the list of event conflicts.

#### DONE (within Event) - update\_person

Change the details of the event to person assignment. Assigning the same person twice is allowed, but will cause a conflict warning. Does not change any existing assignments. The removal of items within the item will cause a change in the item. For example, if role is omitted, any previously set role will be removed.

**Inputs:**

* Required:
  + **scheduler.Event** – the event which is being changed.
  + **person** (see above)

**Outputs:**

* **code** (required) - The code can be looked up using the UserMessages model or None for success.
* **conflicts** (scheduler.Events) – in cases of conflicts, the list of event conflicts.

#### DONE - remove\_person

Remove the person from the event. This is best for cases where the person has withdrawn entirely from the event.

**Inputs:**

* Required:
  + **scheduler.Event** – the event which is being changed.
* Optional (one of the following is required):
  + **User** – identifier for the person booked, if this is used, ANY bookings in this event for this person will be removed. In cases where the person is part of a group, there will be no change – the group will remain booked.

**Outputs:**

* **code** (required) - The code can be looked up using the UserMessages model or None for success.

### Groups

The change here is that an individual is identified by an integer. This would be the profile in the current model. The scheduler shouldn’t have to know that, it just knows that the User is a unique representation of the actual human. The public id is how the person is publicized.

The data within a group is:

* **booking\_id** (required after creation, int) – identifier that uniquely identified the assignment of THIS person, to THIS event, in THIS way. It’s the resource allocation id.
* **group\_id** (required, int) – the private id that is a unique identifier of the GROUP being identified.
* **public\_id** (optional, int) – the public id for something that can be used to publicize this scheduled event. It’s the responsibility of the caller to be sure that this id has an appropriate relationship with the User. In GBE this would be the persona id. If this is not provided, the person’s role and name won’t be listed.
* **people** (1…n) – the people in the group, the list of Users for each member of the group.
* **role** (optional, string) – a descriptor for this person’s relationship to this event.
* **labels** (optional, string) – a list of strings describing the relationship

#### add\_group

Assign a group to the event. Assigning the same person twice is allowed, but will cause a conflict warning. Does not change any existing assignments.

**Inputs:**

* Required:
  + **scheduler.Event** – the event which is being changed.
  + **group** – a single item for the people involved in the event and the relevant details for their participation.
    - **No booking id needs to be provided**.

**Outputs:**

* **code** (required) - The code can be looked up using the UserMessages model or None for success.
* **booking\_id -** only on success. The unique integer referencing this assignment of person to event.
* **conflicts** (scheduler.Events) – in cases of conflicts, the list of event conflicts.

Conflicts are checked for each person in the group.

#### update\_group

Change the details of the event to group assignment. Assigning the same group twice is allowed, but will cause a set of conflict warnings (one for each person). Does not change any existing assignments. The removal of items within the item will cause a change in the item. For example, if role is omitted, any previously set role will be removed.

**Inputs:**

* Required:
  + **scheduler.Event** – the event which is being changed.
  + **Group –** the group to be updated.

**Outputs:**

* **code** (required) - The code can be looked up using the UserMessages model or None for success.
* **conflicts** (scheduler.Events) – in cases of conflicts, the list of event conflicts.

#### remove\_group

Remove assignment of group to the event – in all it’s varieties.

**Inputs:**

* Required:
  + **scheduler.Event** – the event which is being changed.
* Optional (one of the following is required):
  + **group\_id** – integer for the group, if this is used, ANY bookings in this event for this group will be removed. Bookings for an individual person in the group, that are not part of the group bookings will not be removed.

**Outputs:**

* **code** (required) - The code can be looked up using the UserMessages model or None for success.
* **conflicts** (scheduler.Events) – in cases of conflicts, the list of event conflicts.

### Act Management

These could all be done as operations on scheduler.Event. A key difference is that I think that Act should be a description on a type of worker item. I don’t think it should be it’s own thing. I’d be happy with a generic thing to be interpreted by GBE. For example, a “special” value that is always a gbe.Act.id when the role = “Performer”. This interface would still work, but differently.

An ‘act’ structure in scheduler would be a specialized person or group with the typical things listed above PLUS:

* act\_id – a unique act identifier – this is the definition of an act. No act\_id, and it’s not an act and won’t be included in this interface.
* ordering (integer, optional)

#### get\_acts

Get all the acts for an event.

**Inputs:**

* Required:
  + **scheduler.Event** – the event which is being changed.

**Outputs:**

* **code** (required) - The code can be looked up using the UserMessages model or None for success.
* **acts** (list, 0 to many)– the list of acts scheduled for this show.

#### update\_acts

Doing this is bulk because this is how the thread currently works.

An act that isn’t provided will NOT be deleted.

**Inputs:**

* Required:
  + **scheduler.Event** – the event which is being changed.
  + **acts** (list of dict, 0 to many)– the list of acts scheduled for this show.
    - Omitting the person/group information will NOT change it.

**Outputs:**

* **code** (required) - The code can be looked up using the UserMessages model or None for success.
* **conflicts** (scheduler.Events) – in cases of conflicts, the list of event conflicts.

#### add\_act

Assign act to the event. Assigning the same act twice is not allowed, and will cause an error. But a person/group can be assigned to other work within the event.

**Inputs:**

* Required:
  + **scheduler.Event** – the event which is being changed.
  + **act**  – the act scheduled for this event.

**Outputs:**

* **code** (required) - The code can be looked up using the UserMessages model or None for success.
* **booking\_id -** only on success. The unique integer referencing this assignment of person/group to event.
* **conflicts** (scheduler.Events) – in cases of conflicts, the list of event conflicts.

#### remove\_act

Remove act from the event.

**Inputs:**

* Required:
  + **scheduler.Event** – the event which is being changed.
  + **act** (integer) – the act id scheduled for this event.

**Outputs:**

* **code** (required) - The code can be looked up using the UserMessages model or None for success.
* **conflicts** (scheduler.Events) – in cases of conflicts, the list of event conflicts.

## Location Operations

#### get\_locations

Gets a list of currently available Locations.

**Inputs:**

* None

**Outputs:**

* list of locations objects

## Event Evaluation:

For either of these cases, if an occurrence referenced has a start time that is AFTER the current time, the occurrence is not considered valid for evaluation. If this is the only occurrence available, a warning will be returned.

#### get\_eval\_info

Get evaluation info

**Inputs (all are optional):**

* **occurrence\_id** – the unique identifier of the occurrence. If provided, all answers returned are limited to this occurrence.
* **person** – same as the Person listed in Individual Functions. There will be no booking, so at the moment – role and label are irrelevant. If provided, all answers returned will be limited to this person’s answers.
* **visible** – True by default, whether or not to provide no-longer-used questions.
* **labels –** a list of labels for picking occurrences

**Outputs**:

* The typical errors & warnings, if applicable
* **occurrences** = all occurrences that fit the criteria
* **questions** = all applicable questions ordered by order, question items include:
  + **visible** = yes/no on whether they should be displayed to the public
  + **question** = the text of the question
  + **answer\_type** = the format for the answer:
    - **boolean =** true or false
    - **grade** = a report-card like grade (A, B, C, D, F, NA)
    - **comment** = free form text, max 500 chars
* **answers** = all answers that fit the search criteria (occurrence & person). If person and occurrence\_id are provided, and valid – then the absence of answers means the person has not filled out the form before.

#### set\_eval\_info

Sets the evaluation answers provided for a given occurrence by a given person.

* If there is an answer w/out a valid question, then an error will be returned.
* If there are questions w/out answers, no answer will be persisted.
* Blank text and False Booleans **are** stored as answers.

**Inputs (all are required):**

* **answers**:
  + **question** – the question.
  + **value –** a value that fits the answer type. If the value and answer type do not match, an error will be thrown.
* **occurrence\_id** – the unique identifier of the occurrence. If provided, all answers returned are limited to this occurrence.
* **person** – same as the Person listed in Individual Functions. There will be no booking, so at the moment – role and label are irrelevant. If provided, all answers returned will be limited to this person’s answers.

**Outputs**:

* The typical errors & warnings, if applicable
* **occurrence** (required if successful) - The scheduler.Event that was created as a result of this request.
* **answers** = all answers that fit the search criteria (occurrence & person). If person and occurrence\_id are provided, and valid – then the absence of answers means the person has not filled out the form before. Answers have the format:
  + **profile –** submitter’s profile
  + **event –** the occurrence that the eval is about
  + **question –** the question (see questions in get\_eval\_info)
  + **answer –** the actual value (in alignment with the answer type)

## Scheduler Refactoring Needed

### get\_conflicts

isolate the functionality into an all-scheduler version.

### EventItem

Goes away – replaced by an integer. Gbe.Event can receive many of the functions here.

### WorkerItem

Gets two new data items:

* User – required,
* public\_id – optional, integer

Or if it’s a group:

* Set of Users (1:n)
* public\_id

### ActItem

The idea of an Act as a resource forces a blending of the people in the act and the people booked directly for an event. I’d rather use “act” as a decorator to the group/person when the group/person is a performer. This eliminated Act Item and puts it an add-on to WorkerItem.

### Room

Move from gbe to scheduler, I simply can’t see why we would leave this here when it’s not referenced by anything in the model structure here. I’d prefer to collapse Location into Room if it’s not too invasive.

## GBE Refactoring Needed

When people are added or removed from a Troupe, the group should be updated.

Views – virtually all scheduler.views need to be moved to GBE and seriously refactored to use this API. The only candidates that should STAY in scheduler is anything that uses scheduler items **exclusively**.

Templates – need to move with their respective views.

URLs – need to be refactored.

Forms – I’m expecting forms to move largely to gbe. However, if there are forms that are purely scheduler related (especially modelForms), it would be nicer to leave those in scheduler and create a “get form”, and “set form” function in the API.

Strategy – a view by view move, so we can test and work this in parallel with other work. Change the views, templates, URLs, and forms in vertical stacks. A form doesn’t move until all of it’s dependant views move.

# For Later:

I don’t like gbe.Event being stored in scheduler. But that’s a big enough change that I want to get the rest done first.

For parent/child event containers – I’d at least like to do the same – make these not gbe.Events but id’s. Or move to gbe.

Change staff\_areas to a label, not a parent/child.

Staff Areas are not events and should not be treated as such. Scheduling staff areas is both onerous and dumb. They could be treated as Labels quite easily.

Should we refactor roles into labels? In the current DB, they are quite similar. Answer – it’s not that easy. With some labels, it’s important that the role be volunteer. This would be a serious rework of the person assignment process.

Figure out the connection between Rooms and LocationItems – are they in gbe or scheduler? They are still inherited across the structures.

**EventScheduleForm**:

* Could refactor to be the stuff NOT in the various <event>\_schedule\_form
* Could refactor to remove the get\_current\_conference – and instead be seeded by the event that is passed in.
* Could refactor to make the various roles flexible per event type – I’d actually remove them and then make an iterable set of forms.