Planning Results API

Below is a list of provided Results API functions for the planning application. Each function must be called using a Thinknode meta request.

For more information on Thinknode meta requests, please visit the Thinknode documentation.

For an example in calling a Planning Results API function, please refer to the *planning_results_api_example.py* in the .decimal astroid-script-library.

Results API Types

```
// PLAN RESULTS API

// Some general notes about the API...
//
// All API functions here are exposed as request generators. This is for
// multiple reasons...
//
// * It allows the implementation to return (possibly chained) 'meta'
//   requests that retrieve information that's needed to actually get the
//   results.
//
// * It allows users to inspect the intermediate requests to see how result
//   calculations are composed.
//
// 'xx_summary' structures give a light-weight summary of an 'xx'.
// It includes only textual information and IDs. There's no real data.
// The real data is retrieved separately through other API calls.
//
// A 'report' is general information about an object. This can include both
// user commentary and parameters that are automatically captured by the app.
//
// A report has no prescribed structure. It's only meant to be consumed
// visually by humans. The generating app version has full flexibility in what
// it chooses to include. Thus, apps that use this API should NOT analyze the
// contents of the report. If apps need information that's not directly
// available outside the report, then either this API is missing information
// or they're looking for information that's specific to a particular version
// of the planning app (in which case they should be using that version's data
// structures directly).
//
// There's some redundancy in the API (for example, the fact that you can get
// the DVHs for structures even though you have enough info to compute them
```

```
// yourself). This is not just a matter of convenience but is a way of
ensuring
// that important results are captured as they were originally computed.

api(struct)
struct ct_summary
{
    double window;
    double level;
    vector3d slice_positions;
    markup_document report;
};

api(struct)
struct structure_summary
{
    value internal_id;
    // stringified version of the internal id
    string id;
    // display label of the structure
    styled_text label;
    rgb8 color;
    bool visible;
    double opacity;
    structure_render_mode render_mode;
    markup_document report;
};

api(struct)
struct point_summary
{
    value internal_id;
    styled_text label;
    rgb8 color;
    markup_document report;
};

api(struct)
struct aperture_summary
{
    styled_text label;
    markup_document report;

    aperture_parameters parameters;
};

api(struct)
struct degrader_summary
```

```
{
    styled_text label;
    markup_document report;

    double thickness;
};

api(struct)
struct beam_summary
{
    styled_text label;
    rgb8 color;
    markup_document report;

    styled_text isocenter;
    styled_text geometric_target_label;
    styled_text spot_target_label;

    double couch_angle;
    double gantry_angle;

    double air_gap;

    // optional snout name for this beam
    optional<string> snout_name;

    // optional aperture for this beam
    optional<aperture_summary> aperture;

    // list of degraders for this beam
    std::vector<degrader_summary> degraders;
};

api(struct)
struct course_summary
{
    markup_document report;
    styled_text label;
};

api(struct)
struct intent_summary
{
    // a report about what the plan was intended to accomplish
    // (e.g., prescription, goals, etc.)
    markup_document report;
    // Name of the patient external structure (from the disease site template)
    string patient_structure_name;
```

```
    // Name of the treatment_site
    string treatment_site;
};

api(struct)
struct patient_model_summary
{
    markup_document report;
    styled_text label;
};

api(struct)
struct constraint_summary
{
    string constraint_type;
    styled_text structure_label;
    double dose;
};

api(struct)
struct fraction_group_summary
{
    styled_text label;
    rgb8 color;
    markup_document report;

    unsigned fraction_count;
    // The scale factor to apply to the rx_statement for the fraction group.
Set as follows:
    // IMPT Fraction Group: 1.0
    // SFO Fraction Group: 1.0 / # of beamsets
    // Advanced Fraction Group: 1.0
    double rx_scale_factor;

    std::vector<constraint_summary> constraints;

    std::vector<beam_summary> beams;
};


api(struct)
struct prescription_summary
{
    styled_text label;
    rgb8 color;
    markup_document report;

    // the number of fractions this prescription is specified to implement
```

```
    unsigned fraction_count;

    // map of the index of the structure from the structure summary and it's
Rx dose
    std::map<unsigned, double> prescriptions;

    std::vector<fraction_group_summary> fraction_groups;
};

api(struct)
struct spot_placement_summary
{
    double distal_margin;
    double lateral_margin;
    double spot_spacing;
    string spot_strategy;
    double layer_spacing;
    string layer_strategy;
};

api(struct)
struct objective_summary
{
    string objective_type;
    styled_text structure_label;
    optional<double> dose;
};

api(struct)
struct solver_parameter_summary
{
    bool intelligent_stopping_solver;
    unsigned solver_iterations;
    bool intelligent_stopping_optimizer;
    unsigned optimizer_iterations;
};

api(struct)
struct dicom_summary
{
    string study_instance_uid;
    string study_description;
    string study_id;
    string series_instance_uid;
    string series_description;
    string ref_structure_set_uid;
};
```

```
api(struct)
struct rsp_image_summary
{
    markup_document report;
    string rsp_image_id;
};

api(struct)
struct plan_summary
{
    // A summary of the DICOM study/series information
    dicom_summary dcm_summary;

    // the treatment plan name
    styled_text label;

    // the plan description
    styled_text description;

    // the patient information
    // (e.g., name, mrn, demographics)
    patient_info patient;

    // the snapshot patient position
    patient_position_type patient_position;

    // info from the course level
    course_summary course_info;

    // info from the intent level
    intent_summary intent_info;

    // info from the patient model level
    patient_model_summary patient_model_info;

    // a summary about the CT image associated with the treatment plan
    ct_summary ct_image_summary;

    // a report about the high-level creation of the plan
    // (e.g., calculation settings, optimization parameters, etc.)
    markup_document creation_report;

    // a summary about the RSP image associated with the treatment plan
    rsp_image_summary rsp_image_info;

    // a report about the calculation grid used in creating the treatment plan
    markup_document calc_grid_report;
```

```
    // a report about the results of the plan
    // (e.g., dose statistics that were considered important)
    markup_document results_report;

    // the list of structures relevant to this treatment plan
    std::vector<structure_summary> structures;

    // the list of points relevant to this treatment plan
    std::vector<point_summary> points;

    // ID of the machine on which this plan is intended to be treated
    string machine_id;

    // ID of the treatment room on which this plan is intended to be treated
    string treatment_room_id;

    // the list of prescriptions intended to be delivered by this plan
    std::vector<prescription_summary> prescriptions;

    // the spot placement parameters used at the plan level
    spot_placement_summary spot_placement;

    // the list of constraints to be used for calculating MCO in this plan
    std::vector<constraint_summary> constraints;

    // the list of objectives to be used for calculating MCO in this plan
    std::vector<objective_summary> objectives;

    // the iteration counts used for MCO calculations in this plan
    solver_parameter_summary solver_params;
};
```

Results API Functions

```
// Generate the request for the site_info used for constructing the treatment
plan.
api(fun)
calculation_request
generate_site_info_request(treatment_plan const& plan);

// Generate the request for the dicom rt_ion_plan for a treatment_plan
api(fun)
calculation_request
generate_rt_ion_plan_request(
    treatment_plan const& plan,
    rt_approval const& approval,
```

```
    rt_tolerance_table const& tol_table,
    rt_patient_setup const& patient_setup,
    std::vector<rt_dose_reference> const& prescription_data,
    unsigned fraction_cycle_length,
    unsigned fractions_per_day,
    string const& fraction_pattern);

// Generate the request for the summary info for a plan.
api(fun)
calculation_request
generate_plan_summary_request(treatment_plan const& plan);

// Generate the request for the CT image used for constructing a plan.
// Image is returned in IEC 61217 (DICOM) patient space coordinates.
api(fun)
calculation_request
generate_ct_image_request(treatment_plan const& plan);

// Generate the request for the RSP image used for constructing a plan.
// The spatial coordinate system is that of the CT image.
// TODO: This currently returns the merged EPF image because we don't actually
// construct the sliced one as an intermediate step, which is also incorrect.
// Once that's fixed, this should return the sliced EPF.
api(fun)
calculation_request
generate_rsp_image_request(treatment_plan const& plan);

// Generate the request for the machine used for constructing a plan.
api(fun)
calculation_request
generate_machine_request(treatment_plan const& plan);

// Generate the request for the geometry associated with a treatment plan
// structure.
// structure_index is the index of the structure in the plan summary's
// `structures` list.
// The spatial coordinate system of the structure is that of the CT image.
api(fun)
calculation_request
generate_structure_geometry_request(
    treatment_plan const& plan, size_t structure_index);

// Generate the request for the geometry associated with a treatment plan
// point.
api(fun)
calculation_request
generate_point_geometry_request(
    treatment_plan const& plan, size_t point_index);
```

```
// Generate the request for the list of voxels making up the calculation grid
// used for a plan.
api(fun)
calculation_request
generate_calc_grid_voxels_request(
    treatment_plan const& plan);

// Generate the request for the dose delivered by a treatment plan.
// The units of the dose image are Gy(RBE).
// The spatial coordinate system is that of the CT image.
api(fun)
calculation_request
generate_plan_dose_request(treatment_plan const& plan);

// Generate the request for the DVH for a structure within a treatment plan.
// The DVH is cumulative and includes the full plan dose.
// The image-space dimension of the DVH is dose and is in Gy(RBE).
// The value-space dimension of the DVH is volume and is fractional (not
percent).
api(fun)
calculation_request
generate_plan_dvh_request(treatment_plan const& plan, size_t structure_index);

// Generate the request for the dose delivered by a treatment plan for a
// particular prescription.
// The units of the dose image are Gy(RBE).
// The spatial coordinate system is that of the CT image.
api(fun)
calculation_request
generate_prescription_dose_request(
    treatment_plan const& plan,
    size_t prescription_index);

// Generate the request for the dose delivered by a fraction group within a
// treatment plan.
// The fraction group is specified by the index of its prescription within the
plan
// and the index of the fraction group within its prescription.
// The units of the dose image are Gy(RBE).
// The spatial coordinate system is that of the CT image.
api(fun)
calculation_request
generate_fraction_group_dose_request(
    treatment_plan const& plan,
    size_t prescription_index,
    size_t fraction_group_index);
```

```
// Generate the request for the dose delivered by a beam within a treatment
// plan.
// The beam is identified by the index of its prescription within the
prescription, its
// fraction group within that prescription, and finally the actual beam's
index
// within its fraction group.
// The units of the dose image are Gy(RBE).
// The spatial coordinate system is that of the CT image.
api(fun)
calculation_request
generate_beam_dose_request(
    treatment_plan const& plan,
    size_t prescription_index,
    size_t fraction_group_index,
    size_t beam_index);

// Generate the request for the geometry for a beam in a treatment plan.
api(fun)
calculation_request
generate_beam_geometry_request(
    treatment_plan const& plan,
    size_t prescription_index,
    size_t fraction_group_index,
    size_t beam_index);

// Generate the request for the aperture used by a beam within a treatment
// plan.
//
// The generated request will yield an aperture as modeled by the dosimetry
// app.
//
// If the specified beam has no aperture, this is an error.
api(fun)
calculation_request
generate_beam_aperture_request(
    treatment_plan const& plan,
    size_t prescription_index,
    size_t fraction_group_index,
    size_t beam_index);

// Generate the request for the aperture mesh for a beam within a treatment
// plan.
//
// The generated request will yield a 3D mesh that describes the shape that
// the planning app believes the aperture to be. The mesh is oriented in beam
// space with its downstream edge at the isocentric plane.
//
```

```
// If the specified beam has no aperture, this is an error.
api(fun)
calculation_request
generate_beam_aperture_mesh_request(
    treatment_plan const& plan,
    size_t prescription_index,
    size_t fraction_group_index,
    size_t beam_index,
    optional<double> thickness_override);

// Generate the request for a degrader used by a beam within a treatment plan.
api(fun)
calculation_request
generate_beam_degrader_request(
    treatment_plan const& plan,
    size_t prescription_index,
    size_t fraction_group_index,
    size_t beam_index,
    size_t degrader_index);

// Generate the request for the 3D mesh for a degrader used by a beam within a
// treatment plan.
//
// The generated request will yield a 3D mesh that describes the shape that
// the planning app believes the degrader to be. The mesh is oriented in beam
// space with its downstream edge at the isocentric plane.
//
/// TODO: Again, I'm not sure that this should really be a separate request.
///
api(fun)
calculation_request
generate_beam_degrader_mesh_request(
    treatment_plan const& plan,
    size_t prescription_index,
    size_t fraction_group_index,
    size_t beam_index,
    size_t degrader_index);

// Generate the request for the spot list for a beam within a treatment plan.
api(fun)
calculation_request
generate_beam_spot_list_request(
    treatment_plan const& plan,
    size_t prescription_index,
    size_t fraction_group_index,
    size_t beam_index);

// Generate the request for the snout position for a beam within a treatment
```

```
plan.
api(fun)
calculation_request
generate_beam_snout_position_request(
    treatment_plan const& plan,
    size_t prescription_index,
    size_t fraction_group_index,
    size_t beam_index);
```