

Dates and Times

TABLE OF CONTENTS

- [Overview](#)
- [Retrieve a date field](#)
- [Retrieve date range field](#)
- [Formatting date range fields](#)
- [Formatting a date string with an embedded timezone](#)
- [Formatting a date range for display](#)
- [Saving date fields](#)
- [Create DrupalDateTime objects](#)
 - [Create DrupalDateTime objects with timezones](#)
- [Create a DrupalDateTime object and display as a year only](#)
- [Formatting node created time with Drupal date.formatter service](#)
- [Date arithmetic example 1](#)
- [Date arithmetic example 2](#)
- [Comparing DrupalDateTime values](#)
- [Comparing dates \(without comparing times\)](#)
- [Comparing Dates to see if a node has expired](#)
- [Node creation and changed dates](#)
- [Query the creation date using entityQuery](#)
- [Query a date field with no time](#)
- [Query a date field with a time](#)
- [Smart Date](#)
 - [Smart date: Load and format](#)
 - [Smart date: all-day](#)
 - [Smart date: Range of values](#)
- [Reference](#)
 - [Date field storage](#)
 - [DrupalDateTime API reference](#)
 - [UTC](#)
 - [Unix epoch timestamps](#)
 - [Links](#)

views 106

Overview

Drupal Date fields are stored as varchar 20 UTC date strings (e.g. 2022-06-30T12:00:00) while node created and changed fields are stored as int 11 containing Unix epoch timestamps (e.g. 1656379475) in the node_field_data table (fields: created and changed).

Accessing date fields comes in many flavors:

```
// Returns 2021-12-27 for a date only field.
$event_date = $event_node->field_event_date->value;

// Returns 2021-12-28T16:00:00 for a date field with time.
$event_datetime = $event_node->field_event_datetime->value;

// Return a Unix epoch timestamp.
$timestamp = $event_node->field_date->date->getTimestamp();

// Return a formatted date string.
$date_formatted = $event_node->field_date->date->format("Y-m-d H:i:s");
```

Using `$node->field_mydatefield->date` is ideal as it returns a `DrupalDateTime` class which gives you all sorts of goodness including date math capabilities and formatting.

If you need to do calculations involving Unix timestamps, then using `$node->field_mydatefield->getTimestamp()` is useful although `DrupalDateTime` is probably better. More about `DrupalDateTime` at <https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Datetime%21DrupalDateTime.php/class/DrupalDateTime/9.4.x>. Also at <https://drupal.stackexchange.com/questions/252333/how-to-get-formatted-date-string-from-a-datetimeitem-object>

See [Nodes and Fields chapter Date fields section](#) for more on date fields

Retrieve a date field

You can retrieve date fields a few different ways. They are stored as varchar 20 UTC date strings e.g. 2022-06-30T12:00:00

```
// For a date only field, this returns a string like: 2024-08-31.
// For a date field with time, this returns: 2021-12-28T16:00:00.
$end_date = $contract_node->field_contract_date->value;

// Returns unix timestamp e.g. 1725105600
$end_date = $contract_node->field_contract_date->date->getTimestamp();

// Returns a DrupalDateTime object with all its goodness which you can format.
$end_date = $contract_node->field_contract_date->date;
$formatted_date = $end_date->format('m/d/y');
```

Retrieve date range field

To retrieve a date range field from a node, use `value` and `end_value` for the start and end dates:

```
// Magic getters.
$start = $event_node->field_event_date_range->value
$end = $event_node->field_event_date_range->end_value

// Using get().
$start = $event_node->get('field_event_date_range')->value
$end = $event_node->get('field_event_date_range')->end_value

// Using getValue().
$start = $event_node->get('field_event_date_range')->getValue()[0]['value'];
$end = $event_node->get('field_event_date_range')->getValue()[0]['end_value'];
```

Formatting date range fields

Here are two different examples for formatting date fields:

```
// formatted start date
$start_date_formatted = $node->field_date->start_date->format("Y-m-d H:i:s");

// formatted end date
$end_date_formatted = $node->field_date->end_date->format("Y-m-d H:i:s");
```

Use this link at php.net for date format strings <https://www.php.net/manual/en/datetime.format.php#:~:text=format%20parameter%20string-.format.-character>

Formatting a date string with an embedded timezone

Here a date string with an embedded timezone is used to create a `DrupalDateTime` object which is then converted to be stored into a node.

```
use Drupal\Core\DateTime\DrupalDateTime;

$date_string = "2020-08-24T15:28:04+00:00";
$dtdt = new DrupalDateTime($date_string);
$newstring = $dtdt->format("Y-m-d\Th:i:s");
$node->set('field_date', $newstring);
```

Formatting a date range for display

This code shows how to load a date range field from a node. It will ordinarily display like `3/30/2019 - 3/31/2023` however we want it to display like `Mar 30-31, 2023`.

First we retrieve the starting and ending value like this:

```
$from = $node->get('field_date')->getValue()[0]['value'];
$to = $node->get('field_date')->getValue()[0]['end_value'];
```

Here is the entire function as implemented as a `hook_preprocess_node` function in a `.theme` file. We are creating a `scrunch_date` variable to be rendered via a Twig template as shown below:

```

use Drupal\Core\DateTime\DrupalDateTime;

/**
 * Implements hook_preprocess_node
 *
 * @param $variables
 */
function vst_preprocess_node(&$variables) {
  if (!empty($variables['content']['field_date'])) {
    $date = $variables['content']['field_date'];

    $from = new DrupalDateTime($variables['node']->get('field_date')->getValue()[0]['value']);
    $date_array = explode("-", $from);
    $from_day = substr($date_array[2], 0, 2);
    $from_month = $date_array[1];

    $to = new DrupalDateTime($variables['node']->get('field_date')->getValue()[0]['end_value']);
    $date_array = explode("-", $to);
    $to_day = substr($date_array[2], 0, 2);
    $to_month = $date_array[1];

    if ($from_month === $to_month && $from_day !== $to_day) {
      $variables['scrunch_date'] = [
        '#type' => 'markup',
        '#markup' => $from->format("M j-") . $to->format("j, Y"),
      ];
    }

  }

  // For debugging
  // kint($variables);
  // or
  // kpm($variables);
}

```

Now in the twig node template we can output the `scrunch_date` we created.

From `/web/themes/verygood/templates/node/node--seminar--teaser.html.twig`.

```

{% if content.field_date %}
{% if scrunch_date %}
  <div>
    {{ scrunch_date }}
  </div>
{% else %}
  <div>
    {{ content.field_date }}
  </div>
{% endif %}
{% endif %}

```

Saving date fields

Date fields in Drupal are stored as UTC date strings (e.g. 2022-06-30T12:00:00) and when you use `get()` or `set()`, they return strings. If you want to manipulate them, convert them to `DrupalDateTime` objects, then convert them back to strings for saving.

```
$node->set('field_date', '2025-12-31');
$node->set('field_datetime', '2025-12-31T23:59:59');
// Use this for storing created and changed.
$node->set('created', '1760140799');
$node->save();
```

Create DrupalDateTime objects

```
use Drupal\Core\Datetime\DrupalDateTime;

$date = DrupalDateTime::createFromFormat('j-M-Y', '20-Jul-2022');

// Use current date and time
$date = new DrupalDateTime('now');
// Format like Tue, Jul 16, 2022 - 11:34:am
print $date->format('l, F j, Y - H:i');
// OR
// Format like 16-07-2022: 11:43 AM
print $date->format('d-m-Y: H:i A');
```

Create DrupalDateTime objects with timezones

```
// Use current date & time.
$date = new DrupalDateTime();
$date->setTimezone(new \DateTimeZone('America/Chicago'));
// Print current time for the given time zone e.g. 01/23/2023 10:00 pm
print $date->format('m/d/Y g:i a');

// Another variation using specific date and UTC zone
$date = new DrupalDateTime('2019-07-31 11:30:00', 'UTC');
$date->setTimezone(new \DateTimeZone('America/Chicago'));
// prints 07/31/2019 6:30 am
print $date->format('m/d/Y g:i a');
```

UTC: https://en.wikipedia.org/wiki/Coordinated_Universal_Time

Nice article on writing date fields programmatically with more info on UTC timezone at <https://gorannikolovski.com/blog/set-date-field-programmatically#:~:text=Get%20the%20date%20field%20programmatically,%3B%20%2F%2F%20For%20datetime%20fields>.

Create a DrupalDateTime object and display as a year only

This code creates a `Drupal\Core\Datetime\DrupalDateTime` object and returns the year in a render array with some markup. `DrupalDateTime`s are derived from `DateTimePlus` which is a wrapper for PHP `DateTime` class.

```
use Drupal\Core\Datetime\DrupalDateTime;

public function build() {
    $date = new DrupalDateTime();
    return [
        '#markup' => t('Copyright @year&copy; My Company', [
            '@year' => $date->format('Y'),
        ]),
    ];
}
```

Formatting node created time with Drupal date.formatter service

If you want to use a custom date format for your created node date/time you can use one of the methods shown below:

```

$created_date = $node->getCreatedTime();

// Displays 05/04/2022 3:49 pm
$formatted_created_date = \Drupal::service('date.formatter')->format($created_date, 'custom', 'm/d/Y g:i a');

// Displays 2022-05-04 15:49:30
$formatted_created_date = \Drupal::service('date.formatter')->format($created_date, 'custom', 'Y-m-d H:i:s');

// Displays Wed, 05/04/2022 - 15:49
$formatted_created_date = \Drupal::service('date.formatter')->format($created_date);

// Create a DrupalDateTime object and use ->format()
$created_date = $event_node->getCreatedTime();
$cdt = DrupalDateTime::createFromTimestamp($created_date);
$formatted_created_date = $cdt->format('m/d/Y g:i a');

```

See PHP Date format strings: <https://www.php.net/manual/en/datetime.format.php#:~:text=format%20parameter%20string-.format.-character>

Date arithmetic example 1

The code below shows how to add `$days` (an integer) to the date value retrieved from the field: `field_cn_start_date` and save that to the field `field_cn_end_date`.

```

use Drupal\Core\DateTime\DrupalDateTime;

$start_date_val = $node->get('field_cn_start_date')->value;
$days = intval($node->get('field_cn_suspension_length')->value) - 1;
$end_date = DrupalDateTime::createFromFormat('Y-m-d', $start_date_val);
$end_date->modify("+$days days");
$end_date = $end_date->format("Y-m-d");

$node->set('field_cn_end_date', $end_date);
$node->save();

```

Date arithmetic example 2

Here is an example from a module showing a `hook_entity_type_presave()` where some data is changed as the node is being saved. The date arithmetic is pretty simple but the rest of the code is kinda messy.

This is the date arithmetic part:

```

$end_date = DrupalDateTime::createFromFormat('Y-m-d', $start_date_val);
$end_date->modify("+$days days");
$end_date = $end_date->format("Y-m-d");
$node->set('field_cn_end_date', $end_date);

```

The rest of this code does some convoluted wrangling to figure out end dates based on user permissions, changes a node title, looks to see if this is an extension of a previously submitted notice and grabs some date fields from the original notice for use those in the current node.

```

/**
 * Implements hook_ENTITY_TYPE_presave().
 */

function ogg_mods_node_presave(NodeInterface $node) {

  switch ($node->getType()) {

    case 'cat_notice':

      $end_date = NULL != $node->get('field_cn_start_end_dates')->end_value ? $node->get('field_cn_start_end_dates')->end_value : 'n/a';

      $govt_body = NULL != $node->field_cn_governmental_body->value ? $node->field_cn_governmental_body->value : 'Unnamed Government Body';

      $start_date_val = $node->get('field_cn_start_date')->value;

      $accountProxy = \Drupal::currentUser();
      $account = $accountProxy->getAccount();

      // Anonymous users automatically fill out the end_date.
      if (!$account->hasPermission('administer cat notice')) {

        $days = intval($node->get('field_cn_suspension_length')->value) - 1;

        $end_date = DrupalDateTime::createFromFormat('Y-m-d', $start_date_val);
        $end_date->modify("+$days days");
        $end_date = $end_date->format("Y-m-d");
        $node->set('field_cn_end_date', $end_date);
      }

      // Always reset the title.
      $title = substr($govt_body, 0, 200) . " - $start_date_val";
      $node->setTitle($title);

      /*
       * Fill in Initial start and end dates if this is an extension of
       * a previously submitted notice.
       */

      $extension = $node->get('field_cn_extension')->value;
      if ($extension) {

        $previous_notice_nid = $node->get('field_cn_original_notice')->target_id;
        $previous_notice = Node::load($previous_notice_nid);

        if ($previous_notice) {

          $initial_start = $previous_notice->get('field_cn_start_date')->value;
          $initial_end = $previous_notice->get('field_cn_end_date')->value;
          $node->set('field_cn_initial_start_date', $initial_start);
          $node->set('field_cn_initial_end_date', $initial_end);
        }
      }

      break;
    }
  }
}

```

Comparing DrupalDateTime values

The `DrupalDateTime` class extends the `DateTimePlus` class which is a wrapper for PHP `DateTime` class. That functionality allows you to do comparisons. It is probably better manners to use `DrupalDateTime` instead of `DateTime` but here is some `DateTime` code showing how to compare `Datetimes`.

```
date_default_timezone_set('Europe/London');
```

```
$d1 = new DateTime('2008-08-03 14:52:10');
```

```
$d2 = new DateTime('2008-01-03 11:11:10');
```

```
var_dump($d1 == $d2);
```

```
var_dump($d1 > $d2);
```

```
var_dump($d1 < $d2);
```

```
// Returns.
```

```
bool(false)
```

```
bool(true)
```

```
bool(false)
```

Comparing dates (without comparing times)

Use the setTime() function to remove the time part of a datetime so we can make comparisons of just the date.

From a form validation in a .module file.

```
function ogg_mods_cn_form_validate($form, FormStateInterface $form_state) {  
    $start_date = $form_state->getValue("field_cn_start_date");  
    if ($start_date) {  
        $start_date = $start_date[0]['value'];  
        $start_date->setTime(0, 0, 0);  
        $now = new Drupal\Core\DateTime\DrupalDateTime();  
        // Subtract 2 days.  
        $now->modify("-2 days");  
        // Clear the time.  
        $now->setTime(0, 0, 0);  
  
        \Drupal::messenger()->addMessage("Start date = $start_date");  
        \Drupal::messenger()->addMessage("Now date - 2 days = $now");  
  
        if ($start_date < $now) {  
            $form_state->setErrorByName('edit-field-cn-start-date-0-value-date', t('The starting date is more than 2 days in the past. Please select a later date'));  
        }  
    }  
}
```

Comparing Dates to see if a node has expired

This code is used to check if the value in the field field_expiration_date has passed. The field_expiration_date is a standard Drupal date field in a node. In this example, the client wanted to be able to specify the expiration date for nodes.

```
$source_node = $node_storage->load($nid);  
$expiration_date = $source_node->field_expiration_date->value;  
  
// Use expiration date to un-publish expired reseller nodes to hide them.  
$status = 1;  
if ($expiration_date) {  
    $expirationDate = DrupalDateTime::createFromFormat('Y-m-d', $expiration_date);  
    $now = new DrupalDateTime();  
    if ($expiration_date < $now) {  
        // When expired, unpublish the node.  
        $node->set('status', 0);  
        $node->save(); }  
    }  
}
```


It might be interesting to factor in the timezone as date fields are stored in UTC. See https://en.wikipedia.org/wiki/Coordinated_Universal_Time

Node creation and changed dates

Here is a function which does an `entityQuery` for a node and returns a formatted string version of the creation date. Both `created` and `changed` are stored as Unix epoch timestamps in the `node_field_data` table (int 11).

Note. This will handle epoch dates before 1970. You can also use `$node->get('changed')` to retrieve the changed date.

```
use Drupal\Core\Datetime\DrupalDateTime;

protected function loadFirstOpinionYear($term_id) {
  $storage = \Drupal::entityTypeManager()->getStorage('node');
  $query = \Drupal::entityQuery('node')
    ->condition('status', 1)
    ->condition('type', 'opinion')
    ->condition('field_category', $term_id, '=')
    ->sort('title', 'ASC') // or DESC
    ->range(0, 1);
  $nids = $query->execute();
  if ($nids) {
    $node = $storage->load(reset($nids));
  }
  $time = $node->get('created')->value;

  $d = DrupalDateTime::createFromTimestamp($time);
  $str = $d->format("Y-m-d H:i:s");
  return $str;
}
```

To find any nodes that were changed in the last 7 days, use the following:

```
$query = \Drupal::entityQuery('node')
  ->condition('field_tks_program_status', 'ready_for_release')
  ->accessCheck(FALSE)
  ->condition('type', 'teks_pub_program');
// Query changed date within 7 days of today.
$query->condition('changed', strtotime('-7 days'), '>=');

$program_nids = $query->execute();
$program_nids = array_values($program_nids);
```

Query the creation date using entityQuery

This controller example queries for any events for year 2022 with a matching taxonomy term id of 5.

```

public function test2() {

    $str = "Results";
    $year = '2022';
    $term_id = 5;
    $titles = $this->eventsForYear($year, $term_id);
    $count = count($titles);
    $str .= "<br/>Found $count titles for query $year, term_id $term_id";

    foreach ($titles as $title) {
        $str .= "<br/>$title";
    }
    $render_array['content'] = [
        '#type' => 'item',
        '#markup' => $str,
    ];
    return $render_array;
}

private function eventsForYear($year, $term_id): array {
    // Build valid range of start dates/times.
    $format = 'Y-m-d H:i';
    $start_date = DrupalDateTime::createFromFormat($format, $year . "-01-01 00:00");
    $end_date = DrupalDateTime::createFromFormat($format, $year . "-12-31 23:59");

    $start_date_ts = $start_date->getTimestamp();
    $end_date_ts = $end_date->getTimestamp();

    $query = \Drupal::entityQuery('node')
        ->condition('status', 1)
        ->condition('type', 'event')
        ->condition('field_event_category', $term_id, '=')
        ->condition('created', $start_date_ts, '>=')
        ->condition('created', $end_date_ts, '<=')
        ->sort('title', 'DESC');
    $nids = $query->execute();
    $titles = [];
    if ($nids) {
        foreach ($nids as $nid) {
            $node = Node::load($nid);
            $titles[] = $node->getTitle();
        }
    }
    return $titles;
}

```

Read more in the article: Date (range) fields and Entity Query from February 2018 at <https://blog.werk21.de/en/2018/02/05/date-range-fields-and-entity-query-update>

and

this Stack exchange question at <https://drupal.stackexchange.com/questions/198324/how-to-do-a-date-range-entityquery-with-a-date-only-field-in-drupal-8>

Query a date field with no time

Drupal date fields can be configured to store the date and time or only the date. Looking in the database, you might notice that a date-only field is stored like: 2021-12-27 which means you can query just using a string. It probably is wiser to use DrupalDateTime like this:

```

use Drupal\Core\DateTime\DrupalDateTime;
use Drupal\datetime\Plugin\Field\FieldType\DateTimeItemInterface;

public function test4() {

    // Date-only fields are stored in the database like: '2021-12-27';

    // Get a date string suitable for use with entity query.
    $date = DrupalDateTime::createFromFormat('j-M-Y', '27-Dec-2021');
    $date->setTimezone(new \DateTimeZone(DateTimeItemInterface::STORAGE_TIMEZONE));
    // NB. Specify the date-only storage format - not the datetime storage format!
    $query_date = $date->format(DateTimeItemInterface::DATE_STORAGE_FORMAT);

    // Query using =.
    $query = \Drupal::entityQuery('node')
        ->condition('type', 'event')
        ->condition('status', 1)
        ->condition('field_event_date.value', $query_date, '=')
        ->sort('title', 'ASC');
    $nids = $query->execute();
    $count = count($nids);

    $str = "Results";
    $str .= "<br/>Found $count events for field_event_date = $query_date";
    foreach ($nids as $nid) {
        $event_node = Node::load($nid);
        $title = $event_node->getTitle();
        $date = $event_node->field_event_date->value;
        $str .= "<br/>$title - date: $date";
    }
    $str .= "<br/>";

    // Query using >.
    $query = \Drupal::entityQuery('node')
        ->condition('type', 'event')
        ->condition('status', 1)
        ->condition('field_event_date.value', $query_date, '>')
        ->sort('title', 'ASC');
    $nids = $query->execute();
    $count = count($nids);

    $str .= "<br/>Found $count events for field_event_date > $query_date";

    foreach ($nids as $nid) {
        $event_node = Node::load($nid);
        $title = $event_node->getTitle();
        $date = $event_node->field_event_date->value;
        $str .= "<br/>$title - date: $date";
    }

    $render_array['content'] = [
        '#type' => 'item',
        '#markup' => $str,
    ];
    return $render_array;
}

```

Query a date field with a time

```

use Drupal\Core\DateTime\DrupalDateTime;
use Drupal\datetime\Plugin\Field\FieldType\DateTimeItemInterface;

/*
 * Query a date field with a time.
 */

public function test3() {

    // Get a date string suitable for use with entity query.
    $date = new DrupalDateTime();
    // This is a date/time from my local timezone.
    $date = DrupalDateTime::createFromFormat('d-m-Y: H:i A', '28-12-2021: 10:00 AM');
    $date->setTimezone(new \DateTimeZone(DateTimeItemInterface::STORAGE_TIMEZONE));
    $query_date = $date->format(DateTimeItemInterface::DATETIME_STORAGE_FORMAT);

    $str = "Results";

    $query = \Drupal::entityQuery('node')
        ->condition('type', 'event')
        ->condition('status', 1)
        ->condition('field_event_datetime.value', $query_date, '=')
        ->sort('title', 'ASC');
    $nids = $query->execute();
    $count = count($nids);

    $print_date = $date->format('d-m-Y: H:i A');
    $str .= "<br/><strong>$count event(s) for field_event_datetime = $print_date (UTC)</strong> ";
    $date->setTimezone(new \DateTimeZone('America/Chicago'));
    $print_date = $date->format('d-m-Y: H:i A');
    $str .= "<br/>For my timezone (America/Chicago), that is $print_date";

    foreach ($nids as $nid) {
        $event_node = Node::load($nid);
        $title = $event_node->getTitle();
        $display_date = $event_node->field_event_datetime->value;
        $str .= "<br/>$title - date: $display_date";
    }
    $str .= "<br/>";

    $query = \Drupal::entityQuery('node')
        ->condition('type', 'event')
        ->condition('status', 1)
        ->condition('field_event_datetime.value', $query_date, '>')
        ->sort('title', 'ASC');
    $nids = $query->execute();
    $count = count($nids);

    $print_date = $date->format('d-m-Y: H:i A');
    $str .= "<br/><strong>$count event(s) for field_event_datetime > $print_date (UTC)</strong> ";
    $date->setTimezone(new \DateTimeZone('America/Chicago'));
    $print_date = $date->format('d-m-Y: H:i A');
    $str .= "<br/>For my timezone (America/Chicago), that is $print_date";

    foreach ($nids as $nid) {
        $event_node = Node::load($nid);
        $title = $event_node->getTitle();
        $display_date = $event_node->field_event_datetime->value;
        $str .= "<br/>$title - date: $display_date";
    }
}

```

```
$str .= "<br/>";
```

```
$render_array['content'] = [  
  '#type' => 'item',  
  '#markup' => $str,  
];  
return $render_array;  
}
```

More in the article: Date (range) fields and Entity Query from February 2018 at <https://blog.werk21.de/en/2018/02/05/date-range-fields-and-entity-query-update>

and

this Stack exchange question at <https://drupal.stackexchange.com/questions/198324/how-to-do-a-date-range-entityquery-with-a-date-only-field-in-drupal-8>

Smart Date

This module provides the date field that fills in all the gaps in functionality that Drupal core dates lack. Maybe someday it will make it into Drupal core.

This module attempts to provide a more user-friendly date field, by upgrading the functionality of core in several ways:

Easy Admin UI: Includes the concept of duration, so that a field can have a configurable default duration (e.g. 1 hour) and the end time will be auto-populated based on the start. The overall goal is to provide a smart interface for time range/event data entry, more inline with calendar applications which editors will be familiar with.

All Day Events Most calendar applications provide a one-click option to make a an event, appointment, or other time-related content span a full day. This module brings that same capability to Drupal.

Zero Duration Events Show only a single time for events that don't need a duration.

Formatting: More sophisticated output formatting, for example to show the times as a range but with a single output of the date. In the settings a site builder can control how date the ranges will be output, at a very granular level.

Performance: Dates are stored as timestamps to improve performance, especially when filtering or sorting. Concerns with the performance of core's date range have been documented in [#3048072: Date Range field creates very slow queries in Views](#).

Overall, the approach in this module is to leverage core's existing Datetime functionality, using the timestamp storage capability also in core, with some custom Javascript to add intelligence to the admin interface, and a suite of options to ensure dates can be formatted to suit any site's needs.

Display configuration is managed through translatable Smart Date Formats, so your detailed display setup is easily portable between fields, views, and so on. (From https://www.drupal.org/project/smart_date)

Smart date: Load and format

Load the smart date field and use the Drupal date formatting service (`date.formatter`). Smart date fields are always stored as unix timestamp values e.g. 1608566400 which need conversion for human consumption.

```
$start = $node->field_when->value;  
$formatter = \Drupal::service('date.formatter');  
  
//returns something like 12/21/2020 10:00 am  
$start_time = $formatter->format($start, 'custom', 'm/d/Y g:ia');
```

Alternatively, you could load it, create a `DrupalDateTime` and then format it:

```
$start = $node->field_when->value;

$dt = DrupalDateTime::createFromTimestamp($start);

$start_date = $dt->format('m/d/y'); //returns 12/21/22

$start_time = $dt->format('g:ia'); // returns 10:00am
```

Smart date: all-day

To check if a smart date is set to all day, check the duration. If it is 1439, that means all day.

```
$start_ts = $node->field_when->value;

$start_dt = DrupalDateTime::createFromTimestamp($start_ts);

$start_date = $start_dt->format('m/d/Y');

$duration = $node->field_when->duration; //1439 = all day

if ($duration == 1439) {

    $start_time = "all day";

}

else {

    $start_time = $start_dt->format('g:ia');

}
```

Smart date: Range of values

```
//Event start date.

//returns a SmartDateFieldItemList

$whens = $node->get('field_when');

// Each $when is a \Drupal\smart_date\Plugin\Field\FieldType\SmartDateItem.

foreach ($whens as $when) {

    $start = $when->value;

    $end = $when->end_value;

    $duration = $when->duration; //1439 = all day

    $tz = $when->timezone; //"" means default. Uses America/Chicago type format.

}
```

You can also peek into the repeating rule and repeating rule index. These are in the `smart_date_rule` table and I believe the `index` column identifies which item is in the “instances” column.

```
$rrule = $when->rrule;

$rrule_index = $when->rrule_index;
```

Reference

Date field storage

Note. The `node_created` and `changed` fields (int 11) use a Unix epoch timestamp stored in the `node_field_data` table. These have values like 1525302749. Drupal date fields (with times) are stored as UTC strings in varchar 20 fields which look like 2019-05-15T21:32:00.

DrupalDateTime API reference

The `DrupalDateTime` class extends the `DateTimePlus` class which is a wrapper for PHP `DateTime` class. It extends the basic component and adds in Drupal-specific handling, like translation of the `format()` method.

`DateTimePlus` has some static methods to create `DrupalDateTime` objects e.g.

```
DrupalDateTime::createFromArray(['year' => 2010, 'month' => 9, 'day' => 28]);
```

From <https://git.drupalcode.org/project/drupal/-/blob/10.1.x/core/lib/Drupal/Component/Datetime/DateTimePlus.php>:

```

/**
 * Wraps DateTime().
 *
 * This class wraps the PHP DateTime class with more flexible initialization
 * parameters, allowing a date to be created from an existing date object,
 * a timestamp, a string with an unknown format, a string with a known
 * format, or an array of date parts. It also adds an errors array
 * and a __toString() method to the date object.
 *
 * This class is less lenient than the DateTime class. It changes
 * the default behavior for handling date values like '2011-00-00'.
 * The DateTime class would convert that value to '2010-11-30' and report
 * a warning but not an error. This extension treats that as an error.
 *
 * As with the DateTime class, a date object may be created even if it has
 * errors. It has an errors array attached to it that explains what the
 * errors are. This is less disruptive than allowing datetime exceptions
 * to abort processing. The calling script can decide what to do about
 * errors using hasErrors() and getErrors().
 *
 * @method $this add(\DateInterval $interval)
 * @method static array getLastErrors()
 * @method $this modify(string $modify)
 * @method $this setDate(int $year, int $month, int $day)
 * @method $this setISODate(int $year, int $week, int $day = 1)
 * @method $this setTime(int $hour, int $minute, int $second = 0, int $microseconds = 0)
 * @method $this setTimestamp(int $unixtimestamp)
 * @method $this setTimezone(\DateTimeZone $timezone)
 * @method $this sub(\DateInterval $interval)
 * @method int getOffset()
 * @method int getTimestamp()
 * @method \DateTimeZone getTimezone()
 */

```

(More at <https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Datetime%21DrupalDateTime.php/class/DrupalDateTime/9.4.x>)

UTC

Coordinated Universal Time or UTC is the primary time standard by which the world regulates clocks and time. It is within about 1 second of mean solar time at 0° longitude (at the IERS Reference Meridian as the currently used prime meridian) such as UT1 and is not adjusted for daylight saving time. It is effectively a successor to Greenwich Mean Time (GMT). From From

https://en.wikipedia.org/wiki/Coordinated_Universal_Time

Unix epoch timestamps



From <https://www.unixtimestamp.com/> - The unix time stamp is a way to track time as a running total of seconds. This count starts at the Unix Epoch on January 1st, 1970 at UTC. Therefore, the unix time stamp is merely the number of seconds between a particular date and the Unix Epoch. It should also be pointed out (thanks to the comments from visitors to this site) that this point in time technically does not change no matter where you are located on the globe. This is very useful to computer systems for tracking and sorting dated information in dynamic and distributed applications both online and client side.

Links

- Drupal API DrupalDateTime Class
<https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Datetime%21DrupalDateTime.php/class/DrupalDateTime/9.4.x>
- From php.net, the definitive documentation on date format strings
<https://www.php.net/manual/en/datetime.format.php#:~:text=format%20parameter%20string-,format-,character>
- Coordinated Universal Time or UTC Wikipedia article https://en.wikipedia.org/wiki/Coordinated_Universal_Time

- Goran Nikolovski's article: Set date programatically from January 2019 [https://gorannikolovski.com/blog/set-date-field-programmatically#:~:text=Get%20the%20date%20field%20programmatically,\)%3B%20%2F%2F%20For%20datetime%20fields](https://gorannikolovski.com/blog/set-date-field-programmatically#:~:text=Get%20the%20date%20field%20programmatically,)%3B%20%2F%2F%20For%20datetime%20fields)
 - Patrick's article: Date (range) fields and Entity Query from February 2018 <https://blog.werk21.de/en/2018/02/05/date-range-fields-and-entity-query-update>
 - Drupal APIS <https://www.drupal.org/docs/drupal-apis>
-
-

[Back to top](#)

[Drupal at your fingertips](#) by [Selwyn Polit](#) is licensed under [CC BY 4.0](#)  

Page last modified: Apr 13 2023.

[Edit this page on GitHub](#)

This site uses [Just the Docs](#), a documentation theme for Jekyll.