ARCHITECTURE REFERENCE

ODT-MIM

USAREUR-AF

OSDK Maker Package — Ontology Definition Reference

Architecture Reference

HEADQUARTERS
UNITED STATES ARMY EUROPE AND AFRICA
(USAREUR-AF)
Wiesbaden, Germany

20 MARCH 2026

DRAFT — UNOFFICIAL — NOT FOR OPERATIONAL USE

OSDK MAKER PACKAGE — ONTOLOGY DEFINITION REFERENCE

The Maker package provides a type-safe, programmatic way to define ontologies, which are the foundation for structured data in Palantir Foundry. This document provides examples of how to use the maker API to define various ontology entities.

Install:

```
npm install @osdk/maker
```

DEFINING AN ONTOLOGY

An ontology serves as a container for all your type definitions:

```
import { defineOntology } from "@osdk/maker";

await defineOntology("com.example.", async () => {
  // Define your ontology entities here
}, "path/to/output/directory");
```

The namespace parameter (`"com.example."`) prefixes all entity names to avoid naming conflicts.

DEFINING SHARED PROPERTY TYPES (SPTS)

SPTs are reusable property definitions that can be used across different object types and interfaces.

Basic SPTs

```
import { defineSharedPropertyType } from "@osdk/maker";

// Primitive types
const nameProperty = defineSharedPropertyType({
  apiName: "name",
  type: "string",
```

```
  displayName: "Name",
  description: "The name of an entity",
});

const ageProperty = defineSharedPropertyType({
  apiName: "age",
  type: "integer",
  displayName: "Age",
  description: "The age in years",
});

const activeProperty = defineSharedPropertyType({
  apiName: "active",
  type: "boolean",
  displayName: "Active",
  description: "Whether the entity is active",
});

const birthdateProperty = defineSharedPropertyType({
  apiName: "birthdate",
  type: "date",
  displayName: "Birth Date",
  description: "Date of birth",
});

const updatedAtProperty = defineSharedPropertyType({
  apiName: "updatedAt",
  type: "timestamp",
  displayName: "Last Updated",
  description: "When the entity was last updated",
});
```

Array SPT

```
const tagsProperty = defineSharedPropertyType({
  apiName: "tags",
  type: "string",
  array: true,
  displayName: "Tags",
  description: "List of tags",
});
```

Struct SPT

```
const addressProperty = defineSharedPropertyType({
  apiName: "address",
  type: {
    type: "struct",
    structDefinition: {
      street: "string",
      city: "string",
```

```
      state: "string",
      zipCode: {
        fieldType: "string",
        displayMetadata: {
          displayName: "ZIP Code",
          description: "The postal code",
        },
      },
      country: "string",
    },
  },
  displayName: "Address",
  description: "Physical address information",
});
```

Specialized String SPT

```
const descriptionProperty = defineSharedPropertyType({
  apiName: "description",
  type: {
    type: "string",
    isLongText: true,
    supportsEfficientLeadingWildcard: true,
    supportsExactMatching: false,
  },
  displayName: "Description",
  description: "Detailed description text",
});
```

Security Marking

```
const securityMarkingProperty = defineSharedPropertyType({
  apiName: "securityMarking",
  type: {
    type: "marking",
    markingType: "CBAC",
  },
  displayName: "Security Marking",
  description: "Security classification",
});
```

Geographic Data

```
const locationProperty = defineSharedPropertyType({
  apiName: "location",
  type: "geopoint",
  displayName: "Location",
  description: "Geographic coordinates",
});
```

```
const areaProperty = defineSharedPropertyType({
  apiName: "area",
  type: "geoshape",
  displayName: "Area",
  description: "Geographic area or region",
});
```

DEFINING VALUE TYPES

Value types allow you to define custom data types with specific constraints and validation rules.

```
import { defineValueType } from "@osdk/maker";

// String value type with regex constraint
defineValueType({
  apiName: "emailAddress",
  displayName: "Email Address",
  type: {
    type: "string",
    constraints: [{
      constraint: {
        regex: "^[\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$",
      },
      failureMessage: "Must be a valid email address",
    }],
  },
  version: "1.0.0",
});

// Boolean value type with allowed values
defineValueType({
  apiName: "trueFalseValue",
  displayName: "True/False Value",
  type: {
    type: "boolean",
    constraints: [{
      constraint: {
        allowedValues: ["TRUE_VALUE"],
      },
    }],
  },
  version: "0.1.0",
});
```

DEFINING INTERFACES

Interfaces define a contract that objects can implement, specifying a set of properties that must be present.

Basic Interface

```
import { defineInterface } from "@osdk/maker";

const personInterface = defineInterface({
  apiName: "Person",
  displayName: "Person",
  description: "Represents a person",
  properties: {
    firstName: "string",
    lastName: "string",
    email: "string",
    age: "integer",
  },
});

// Using existing SPTs
const employeeInterface = defineInterface({
  apiName: "Employee",
  displayName: "Employee",
  description: "Represents an employee",
  properties: {
    firstName: nameProperty,
    lastName: nameProperty,
    employeeId: "string",
    department: "string",
    hireDate: "date",
    isActive: "boolean",
  },
});
```

Optional Properties

```
const customerInterface = defineInterface({
  apiName: "Customer",
  displayName: "Customer",
  properties: {
    name: "string",
    email: "string",
    phoneNumber: {
      required: false,
      propertyDefinition: "string",
    },
```

```
  },
});
```

Interface Extension

```
const managerInterface = defineInterface({
  apiName: "Manager",
  displayName: "Manager",
  properties: {
    managementLevel: "string",
    directReports: "integer",
  },
  extends: [employeeInterface],
});

const executiveInterface = defineInterface({
  apiName: "Executive",
  displayName: "Executive",
  properties: {
    stockOptions: "boolean",
  },
  extends: ["Manager"],  // Extend by apiName
});
```

Interface Status

```
const legacyInterface = defineInterface({
  apiName: "LegacySystem",
  displayName: "Legacy System",
  properties: {
    systemName: "string",
    version: "string",
  },
  status: {
    type: "deprecated",
    message: "This interface is being phased out",
    deadline: "2026-01-01T00:00:00.000Z",
  },
});

const experimentalInterface = defineInterface({
  apiName: "ExperimentalFeature",
  displayName: "Experimental Feature",
  properties: {
    featureName: "string",
    enabled: "boolean",
  },
  status: {
    type: "experimental",
```

```
  },
});
```

DEFINING OBJECTS

Objects represent the core data entities in your ontology.

Basic Object

```
import { defineObject } from "@osdk/maker";

const personObject = defineObject({
  apiName: "person",
  displayName: "Person",
  pluralDisplayName: "People",
  titlePropertyApiName: "name",
  primaryKeyPropertyApiName: "id",
  properties: {
    "id": { type: "string", displayName: "ID" },
    "name": { type: "string" },
    "email": { type: "string" },
    "age": { type: "integer" },
  },
});
```

Object Implementing Interface

```
const employeeObject = defineObject({
  apiName: "employee",
  displayName: "Employee",
  pluralDisplayName: "Employees",
  titlePropertyApiName: "name",
  primaryKeyPropertyApiName: "id",
  properties: {
    "id": { type: "string", displayName: "ID" },
    "name": { type: "string" },
    "email": { type: "string" },
    "department": { type: "string" },
    "hireDate": { type: "date", displayName: "Hire Date" },
    "isActive": { type: "boolean" },
  },
  implementsInterfaces: [
    {
      implements: employeeInterface,
      propertyMapping: [
        { interfaceProperty: "firstName", mapsTo: "name" },
```

```
        { interfaceProperty: "lastName", mapsTo: "name" },
        { interfaceProperty: "employeeId", mapsTo: "id" },
        { interfaceProperty: "department", mapsTo: "department" },
        { interfaceProperty: "hireDate", mapsTo: "hireDate" },
        { interfaceProperty: "isActive", mapsTo: "isActive" },
      ],
    },
  ],
});
```

Stream-Backed Object

```
const eventObject = defineObject({
  apiName: "event",
  displayName: "Event",
  pluralDisplayName: "Events",
  titlePropertyApiName: "eventName",
  primaryKeyPropertyApiName: "eventId",
  properties: {
    "eventId": { type: "string", displayName: "Event ID" },
    "eventName": { type: "string", displayName: "Event Name" },
    "timestamp": { type: "timestamp" },
  },
  datasources: [{
    type: "stream",
    retentionPeriod: "P90D",  // 90 days (ISO 8601 duration format)
  }],
});
```

DEFINING LINKS

Links define relationships between objects.

One-to-Many Link

```
import { defineLink } from "@osdk/maker";

const departmentToEmployeesLink = defineLink({
  apiName: "departmentToEmployees",
  one: {
    object: departmentObject,
    metadata: {
      apiName: "employees",
      displayName: "Employee",
      pluralDisplayName: "Employees",
      visibility: "NORMAL",
```

```
    },
  },
  toMany: {
    object: employeeObject,
    metadata: {
      apiName: "department",
      displayName: "Department",
      pluralDisplayName: "Departments",
      visibility: "NORMAL",
    },
  },
  manyForeignKeyProperty: "departmentId",
});
```

Many-to-Many Link

```
const productToCategoriesLink = defineLink({
  apiName: "productToCategories",
  many: {
    object: productObject,
    metadata: {
      apiName: "categories",
      displayName: "Category",
      pluralDisplayName: "Categories",
      visibility: "NORMAL",
    },
  },
  toMany: {
    object: categoryObject,
    metadata: {
      apiName: "products",
      displayName: "Product",
      pluralDisplayName: "Products",
      visibility: "NORMAL",
    },
  },
});
```

Intermediary (Object-Backed) Link

```
// Define an object-backed link type via an intermediary object
const aircraftToFlightsLink = defineLink({
  apiName: "aircraftToFlights",
  many: {
    object: aircraft,
    metadata: {
      displayName: "Flight",
      pluralDisplayName: "Flights",
      apiName: "flights",
    },
    linkToIntermediary: aircraftToManifestLink,
```

```
  },
  toMany: {
    object: flight,
    metadata: {
      displayName: "Aircraft",
      pluralDisplayName: "Aircraft",
      apiName: "aircraft",
    },
    linkToIntermediary: flightsToManifestLink,
  },
  intermediaryObjectType: manifest,
});
```

INTERFACE LINK CONSTRAINTS

Interface Link Constraints define relationships between interfaces.

```
import { defineInterfaceLinkConstraint } from "@osdk/maker";

const managerToEmployeesLink = defineInterfaceLinkConstraint({
  apiName: "managerToEmployees",
  from: managerInterface,
  toMany: employeeInterface,
});

const personToAddressLink = defineInterfaceLinkConstraint({
  apiName: "personToAddress",
  from: personInterface,
  toOne: addressInterface,
});
```

DEFINING ACTIONS

CRUD Actions

```
import {
  defineCreateObjectAction,
  defineModifyObjectAction,
  defineDeleteObjectAction,
} from "@osdk/maker";

const createEmployeeAction = defineCreateObjectAction({
  objectType: employeeObject,
```

```
});

const modifyEmployeeAction = defineModifyObjectAction({
  objectType: employeeObject,
});

const deleteEmployeeAction = defineDeleteObjectAction({
  objectType: employeeObject,
});
```

Interface Actions

```
import {
  defineCreateInterfaceObjectAction,
  defineModifyInterfaceObjectAction,
} from "@osdk/maker";

const createPersonAction = defineCreateInterfaceObjectAction({
  interfaceType: personInterface,
});

const modifyPersonAction = defineModifyInterfaceObjectAction({
  interfaceType: personInterface,
  excludedProperties: ["primaryKey"],
});
```

ADVANCED: DERIVED PROPERTIES

Objects can have derived properties computed at runtime from linked objects:

```
const flight = defineObject({
  displayName: "Flight",
  pluralDisplayName: "Flights",
  apiName: "flight",
  primaryKeyPropertyApiName: "id",
  titlePropertyApiName: "id",
  properties: {
    id: { type: "string", displayName: "ID" },
    passengersList: {
      type: "string",
      array: true,
      displayName: "Passengers",
    },
  },
  datasources: [
    { type: "dataset" },
    {
      type: "derived",
```

```
      linkDefinition: [{ linkType: flightToPassengers }],
      propertyMapping: {
        passengersList: {
          type: "collectList",
          property: "name",
          limit: 100,
        },
      },
    },
  ],
});
```

SEE ALSO

- MIM_ACADEMICS.md — MIM→Foundry structural alignment analysis
- MIM_OVERVIEW.md — MIM toolchain overview and package structure
- CDA Agents: Ontology Engineer — CDA ontology engineering doctrine