API's –
Accessing the Encyclopedia

Session 140

Frances Casey
Texas Instruments

Composer Encyclopedia API

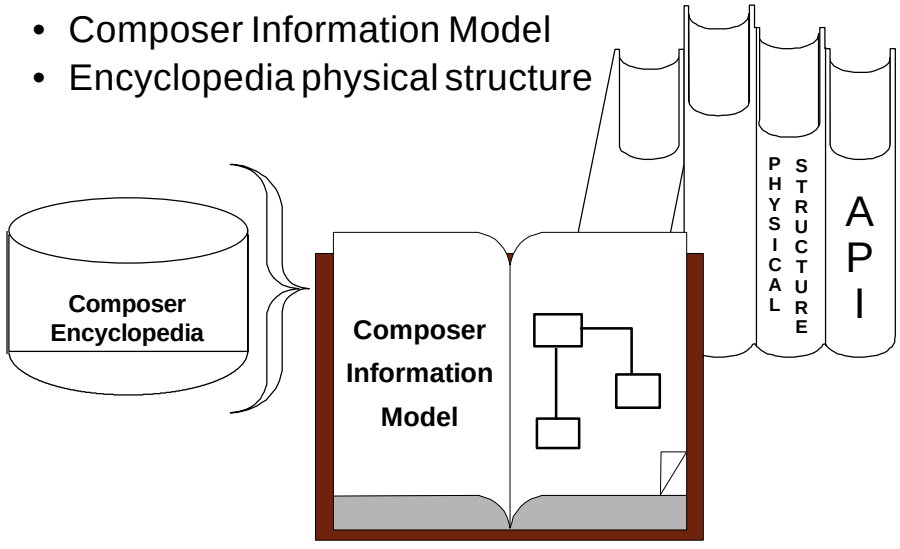
- Introduction to the Composer Open Initiative
- How Composer works
- How model information is stored
- Reading Composer information using the Application Programming Interfaces (APIs)
- Tips on getting started

Composer Open Initiative

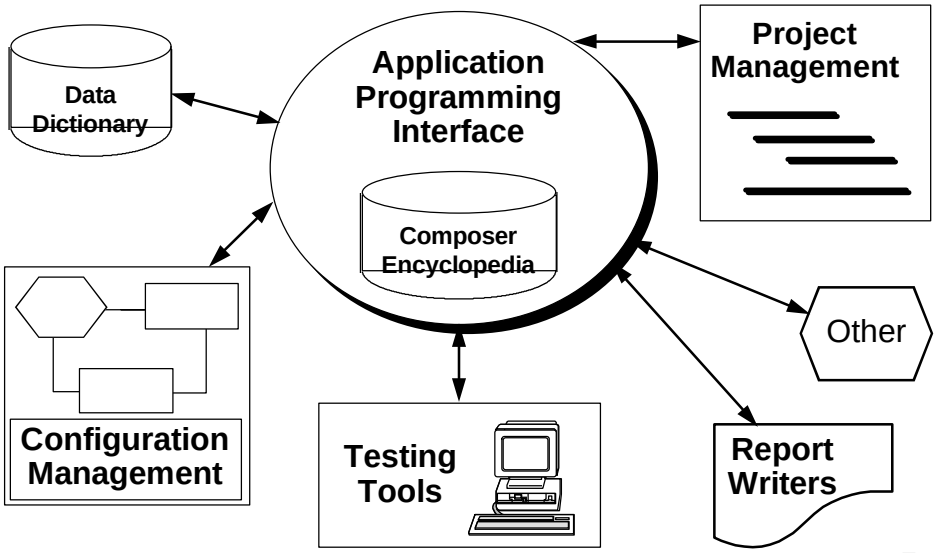
- Published architectures
  - Development tools
  - Application runtime
- Application Programming Interfaces (APIs)
- Alliances for complementary products

Technical Documentation

- Composer Information Model
- Encyclopedia physical structure



Composer Encyclopedia APIs



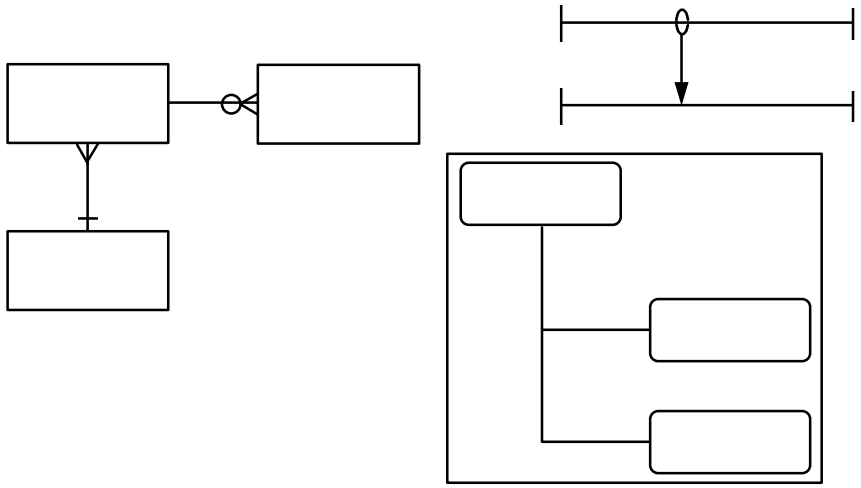
Using the API

- To effectively use the Composer Encyclopedia Application Programming Interface (API), you need to understand how:
  - Composer works
  - Composer model information is stored
  - to read Composer information using APIs
  - to program in C or C++

How Composer Works

Diagrams to Code

- Composer toolkit diagrams support analysis and design of business information requirements in terms of:
  - Data
  - Activities
  - Interaction
- Rigorous diagramming conventions convey specific meaning to each graphic representation



Encyclopedia

- The encyclopedia is the heart of Composer model-driven development strategy
  - Stores all information
  - Controls access for authorization and concurrent development
  - Supports multi-user development by model
  - Supports multi-project development with separate models
  - Primary support for application generation

Encyclopedia Architecture

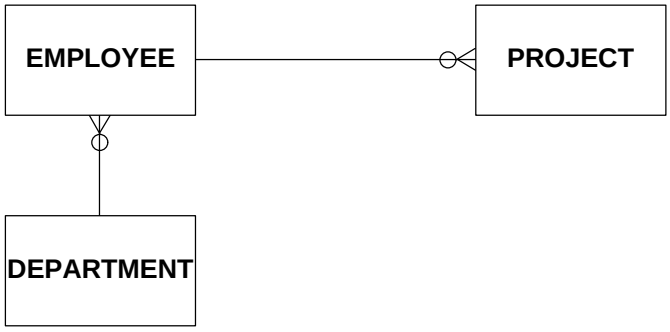
- Developed using the same concepts used for Composer toolset diagrams
  - A model of the process of building models
  - Data about data
- Consists of a set of relational tables
- A data management and reporting tool for data collected during systems development

How Composer Model Information is Structured and Stored

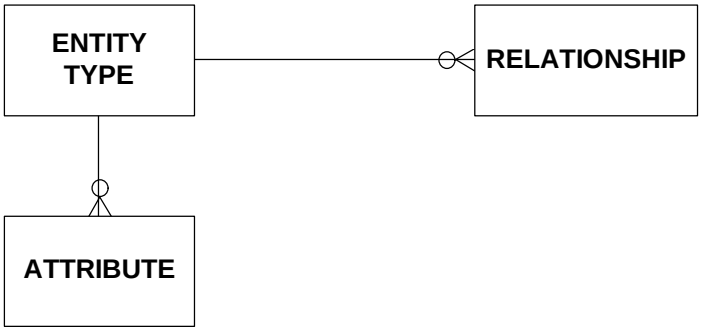
Model, Meta-Model, Meta-Meta-Model

- A Composer *MODEL* forms a picture, or representation, of the business of interest
- The Composer *META-MODEL* can be viewed as the data model of the modeling process
- The Composer *META-META-MODEL* represents how the *META-MODEL* is implemented

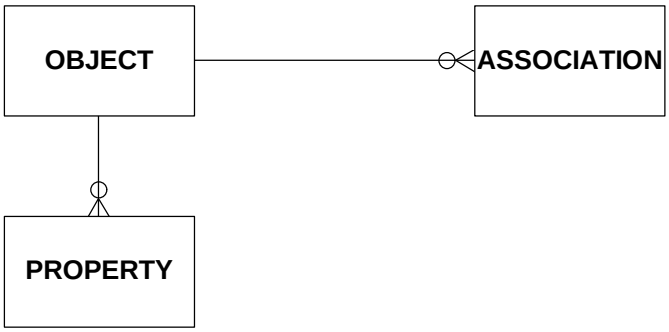
Model



Meta-Model



Meta-Meta-Model



Schema Release Levels

- The schema number defines the meta-model version used for a particular software release
- All Composer tools at the same release level use the same schema
  - Workstation toolsets
  - Client/Server Encyclopedias
  - Host Encyclopedia
- The Encyclopedia APIs apply to all encyclopedia platforms

Summary

- Information is entered as diagrams; stored as semantic meaning
- Rigorous diagramming conventions provide the precision necessary to automatically generate application code from diagrams
- The encyclopedia is the heart of this model-driven development strategy

Using the Encyclopedia API

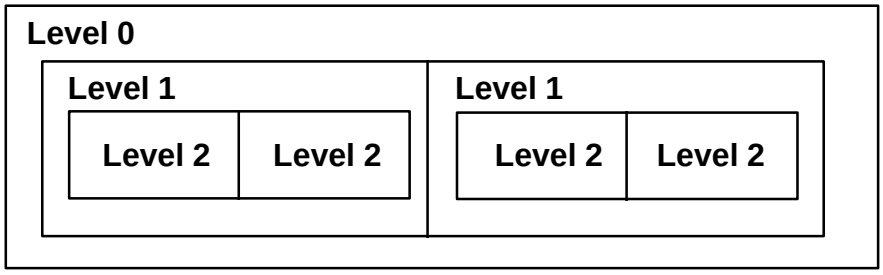
Encyclopedia Tools

- Encyclopedia tools available to view the meta-model structure
  - Public Interface view definitions
  - Object Decomposition Report (DECOMP)
  - Orchestra
  - WalkEncy
  - API functions
  - Schema tables

Object Decomposition Report

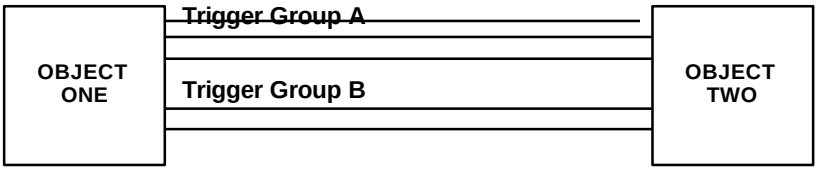
- Full meta-model documentation in report format
- Hierarchical physical structure including objects, properties, associations, and triggers
- Supplemented with indices of objects, properties, and associations by short name (mnemonic) and by long name
- Includes more than is currently used
  - objects, properties, associations defined for unimplemented functionality
  - future capabilities

Hierarchical Structure



- Each level inherits all information from levels above it
- Only the lowest level objects are stored
- Think ‘subtypes’

Trigger Groups



- An association may belong to a Trigger Group, which further defines conditional optionality
- Generally, at least one association within the Group must be present for the object to exist

Object Decomposition Symbols

(#)	Object hierarchy values, object, property or association mnemonics, or trigger association counts
**>	Object meta-properties
---	Object properties
..>	Forward associations
..<	Backward associations
==>	Trigger delete association groups
-->	Physical structure groups
type	Property types (USHORT, SHORT, ULONG, LONG, CHAR, NAME, MACRONAME, LOADNAME, STRING, DESC)
req, opt	Required or optional property
dflt=	Default property values
[ ]	Protection indicators used by Subsetting (modifying, referencing)
{ }	Aggregate action indicators used by Version Control (copy, include, required, optional, special, ignore, notused)

Object Decomposition Example

```
| (4) Identifier (IDENT)
| **>Boundary
| **>Has ART
 ---Primary Identifier? Y/N/sp (PRIMARY), type CHAR, opt, dflt=space, column=2
|..>sometimes contains (CNTNSA, inv INIDTA) many ATTR [modifying] {required}
|..>sometimes contains (CNTNSR, inv INIDTR) many RELMM [modifying] {required}
 ..> sometimes is implemented (IMPLNTBY, inv IMPLMNTI) many ENTRYPNT
     [modifying] {ignore}
 ..> sometimes used as target of fk (TARGETOF, inv TARGETS) many LINKFK
     [modifying] {ignore}
| ..> sometimes contains (CNTNSH, inv INIDTH) many ATTRINH [modifying] {copy}
 ..< always identifies (IDENTS, inv IDNTBY) one ENTY [modifying] {required}
 ==> trigger assoc CNTNSA (1), in group B
 ==> trigger assoc IDENTS (1), in group A
 ==> trigger assoc CNTNSR (1), in group B
| --> physical structure group “5”
```

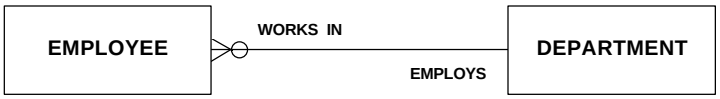

Orchestra – Online Decomp

- Composer Information Model 6.0.C7 - Hierarchy
- Composer Information Model 6.0.C7 - Collapsed
  - Hypertext help versions of Object Decomposition Report
  - Hierarchy version shows properties and associations at the levels in the hierarchy at which they are defined
  - Collapsed version shows properties and associations directly or indirectly inherited by the object type

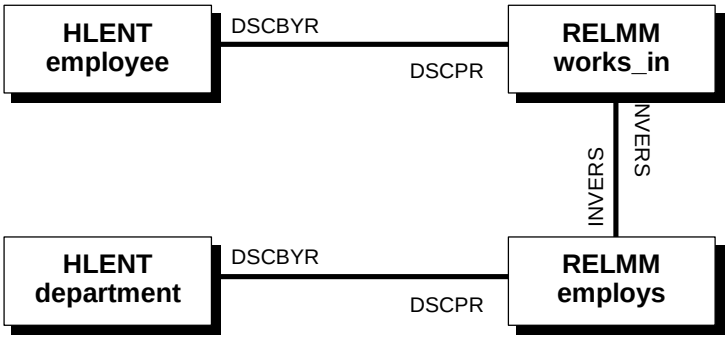
Translating Diagrams

- To write a program accessing model information, you must be able to specify the program in terms of the Information Model
- Objects, properties, and associations are stored with a numeric type code that represents the mnemonic
- Each model is uniquely identified with a numeric model_id; each object is uniquely identified with a numeric object_id
  - Names are properties

Data Model Example



Occurrence Diagram



Objects

OBJ_ID	OBJ_MODEL_ID	OBJ_TYPE_CODE
10	33	113*
11	33	113
12	33	51**
13	33	51

*HLENT

**RELMM

Associations

ASSOC_ FROM_OBJ_ID	ASSOC_ TYPE_CODE	ASSOC_ TO_OBJ_ID
10	70*	12
11	70	13
12	186**	13

*DSCBYR

**INVERS

Properties

PROP_ OBJ_ID	PROP_ TYPE_CODE	PROP_ CHAR_VALUE
10	224*	EMPLOYEE
11	224	DEPARTMENT
12	224	WORKS_IN
13	224	EMPLOYS

* NAME

Property Type Codes are not stored on CSE

Encyclopedia API Functions

Encyclopedia API

- Read-only C subroutines to access schema, model, and administrative information
- Same function call for Host or CSE
- Provided as a static library or dynamic link library (OS/2 only)
- Header files and example programs also provided
- Encyclopedia must be active to run

Why API?

- Protected, interpretive access to model information
- Portability between platforms
- Model lock/unlock during access
- Future expansion of functionality

API Function Types

- API administrative functions control access to the encyclopedia
- Count functions determine number of occurrences
- Array functions retrieve occurrences
  - Allocate memory for the array before call
  - Maximum number of occurrences is input parameter
  - Result is lesser of number of occurrences specified or number in encyclopedia

API Function Categories

- Encyclopedia Information
- Model Lock/Unlock
- Model Information
- Association Information
- Property Information
- Subset Information
- Checkout Information
- User/Group Information
- Authorization Information
- Schema Information

Referencing the Meta-Model

- Headers provided with Encyclopedia API support using mnemonics rather than numeric codes for object, property, and association types
- If you use APIs from non-C program, you will need to use the numeric codes

Example: List All Entities in Model

- Program specification
  - Connect to encyclopedia ‘DBIEFD’
  - Logon to encyclopedia as user ‘DAACME’
  - Fetch the id for a model, using the model name ‘COMPOSER 3 TEST’
  - Lock the model
  - Count the number of entity types
  - Fetch the entity types
  - Commit to release database locks
  - Unlock the model
  - Disconnect from the encyclopedia

Example: List All Entities in Model

```
#include <eapidef.h>      /* API header file                          */
#include <otc.h>          /* Mnemonics for Object Type Codes  */
/* Use API variable types to define parameters  */
EAPIRC              rc;
DBPARMS             szDBParms;
USERID              szUserid;
NAME                szModelName;
MODELID             nModelId;
ENCYLOCKTYPE        eLockType;
LCOUNT              lCount;
OBJID               * hlentids;
```


Example: List All Entities in Model

```
strcpy(szDBParms, “DBNAME=DBIEFD DBUSER=     DBPSWD= ”);
rc = EApiConnectToEncy(szDBParms);

strcpy(szUserid, “DAACME”);
rc = EApiLogonUserId(szUserId);

strcpy(szModelName, “COMPOSER 3 TEST”);
rc = EApiFetchModelByName(szModelName, &nModelId)

eLockType = EAPI_READ_LOCK;
rc = EApiLockModel(nModelId, eLockType);

rc = EApiCountModelTypeObjs(nModelId, OTC_HLENT, &lCount);
```


Example: List All Entities in Model

```
/*  Allocate space for entity type array          */
hlentids = (OBJID *)calloc((size_t)lcount, sizeof(OBJID));

rc = EApiFetchModelTypeObjs(nModelId, OTC_HLENT,
   &lcount, hlentids);

rc = EApiCommit();

rc = EApiUnLockModel(nModelId);

rc = EApiDisconnectEncy();
```


API Future Plans

Composer 4

- Scheduled for 4Q96
- Will include Phase II Encyclopedia API functionality
  - Remote Access
  - Access to workstation encyclopedia

API's –
Accessing the Encyclopedia

Session 140

Frances Casey
Texas Instruments