💻 Complete Code Explanation - Part 6

Backend (IRIS ObjectScript) - Database Classes

Introduction to InterSystems IRIS & ObjectScript

Before diving into the code, let me explain what IRIS and ObjectScript are:

What is InterSystems IRIS?

IRIS = **I**ntersystems **R**elational **I**ntelligent **S**ystem

- **Multi-model database**: Supports objects, SQL, key-value, documents
- **High performance**: Used in healthcare, banking, government
- **Built-in web services**: REST APIs, SOAP, etc.
- **Object persistence**: Classes directly map to database tables

What is ObjectScript?

- **Native language** for IRIS
- **Object-oriented**: Classes, inheritance, methods
- **Database integration**: Direct database operations
- **Syntax**: Mix of BASIC, Pascal, and C-like syntax

Key Syntax Differences from JavaScript:

Feature	JavaScript	ObjectScript
Class keyword	`class`	`Class`
Method	`function method()`	`ClassMethod Method()`
Variable	`let x = 5`	`Set x = 5`
If statement	`if (x === 5)`	`If x = 5`
String concat	`"Hello " + name`	`"Hello "_name`
Comment	`// comment`	`// comment`
This	`this.name`	`..Name`
New object	`new MyClass()`	`##class(MyClass).%New()`

17. SEM.tblAccount.cls

Purpose: User account database class। Signup করা users এর তথ্য store করে।

```objectscript
```

```
/// User account table for authentication
Class SEM.tblAccount Extends %Persistent
{

/// Email address (unique identifier)
Property Email As %String(MAXLEN = 32) [ Required ];

/// User password (should be hashed in production)
Property Password As %String(MAXLEN = 32);

/// User's full name
Property Name As %String(MAXLEN = 64);

/// Ensure email is unique
Index EmailIndex On Email [ Unique ];

}
```

Line-by-Line Explanation:

Line 1-2: Class Definition

```objectscript
/// User account table for authentication
Class SEM.tblAccount Extends %Persistent
```

- `///`: Documentation comment (three slashes)
- `Class SEM.tblAccount`: Class name
  - **SEM**: Package name (namespace)

    - **tblAccount**: Class name (table name)
- `Extends %Persistent`: Inherits from persistent base class
    - **%Persistent**: Built-in IRIS class for database objects
    - Makes this class persistent (saved to database)

What does Extends %Persistent do?

```objectscript
// Without %Persistent:
Class MyClass {
  Property Name;
}
// Just a regular class, data not saved

// With %Persistent:
Class MyClass Extends %Persistent {
  Property Name;
}
// Automatically:
// - Creates database table
// - Provides %Save() method
// - Provides %OpenId() method
// - Provides %DeleteId() method
// - Generates SQL queries
```

Line 3: Opening Brace

```objectscript
```

```
{
```

- Class body starts

Line 5-6: Email Property

```objectscript
/// Email address (unique identifier)
Property Email As %String(MAXLEN = 32) [ Required ];
```

Breaking it down:

- `Property`: Declares a class property (like a field/column)
- `Email`: Property name
- `As %String`: Data type
- `(MAXLEN = 32)`: Maximum length parameter
- `[ Required ]`: Property keyword (validation)
- `;`: Statement terminator

Data Types in ObjectScript:

```objectscript
```

```
%String       → VARCHAR (text)
%Integer      → INTEGER (whole numbers)
%Boolean      → BIT (true/false)
%DateTime     → DATETIME (date and time)
%Date         → DATE (date only)
%Numeric      → DECIMAL (precise numbers)
```

Required Keyword:

```objectscript
// With Required:
Property Email [ Required ];
// Cannot save object if Email is empty
// Database enforces NOT NULL

// Without Required:
Property Email;
// Can be NULL in database
```

Line 8-9: Password Property

```objectscript
/// User password (should be hashed in production)
Property Password As %String(MAXLEN = 32);
```

- Similar to Email
- **No `[Required]`**: Optional (but should be required in practice)
- **Comment warns**: Should be hashed (bcrypt, etc.)

Security Note:

```objectscript
// ❌ Current (BAD - stores plain text):
Property Password As %String(MAXLEN = 32);

// ✅ Production should be (GOOD - hashed):
Property PasswordHash As %String(MAXLEN = 60);
// Store bcrypt hash, not plain password
// Hash length: 60 chars for bcrypt
```

Line 11-12: Name Property

```objectscript
/// User's full name
Property Name As %String(MAXLEN = 64);
```

- User's display name
- Up to 64 characters
- Supports Japanese characters (UTF-8)

Line 14-15: Unique Index

```objectscript
/// Ensure email is unique
Index EmailIndex On Email [ Unique ];
```

What is an Index?

- **Index**: Database structure for fast lookups
- **Unique**: No two rows can have same email
- **EmailIndex**: Name of the index

Why Unique Index?

```objectscript
// Without Unique Index:
// User can sign up multiple times with same email
Email: "user@example.com"  → OK
Email: "user@example.com"  → OK (duplicate!)

// With Unique Index:
Email: "user@example.com"  → OK
Email: "user@example.com"  → ERROR! Duplicate key
```

Database Level:

```sql
-- IRIS automatically creates:
CREATE UNIQUE INDEX EmailIndex ON SEM.tblAccount (Email);

-- Ensures:
-- 1. Fast lookup by email (O(log n) instead of O(n))
-- 2. No duplicate emails possible
-- 3. Email validation at database level
```

Line 17: Closing Brace

```objectscript
}
```

- Class definition ends

Generated Database Table:

When this class is compiled, IRIS automatically creates:

```sql
-- SQL Table (automatically generated):
CREATE TABLE SEM.tblAccount (
  ID INTEGER PRIMARY KEY AUTO_INCREMENT,
  Email VARCHAR(32) NOT NULL,
  Password VARCHAR(32),
  Name VARCHAR(64)
);

-- Unique Index:
CREATE UNIQUE INDEX EmailIndex ON SEM.tblAccount (Email);
```

IRIS Magic:

- **ID**: Automatically added (primary key)
- **Column types**: Derived from Property types
- **NOT NULL**: From [Required] keyword

- **Index**: From Index definition

Using This Class in Code:

Create new account:

```objectscript
// Create new object
Set account = ##class(SEM.tblAccount).%New()

// Set properties
Set account.Email = "user@example.com"
Set account.Password = "password123"
Set account.Name = "田中太郎"

// Save to database
Set status = account.%Save()

// Check if successful
If $$$ISOK(status) {
  Write "Account created with ID: ", account.%Id()
} Else {
  Write "Error: ", $System.Status.GetErrorText(status)
}
```

Find account by email:

```objectscript
```

```
// SQL query
Set sql = "SELECT ID FROM SEM.tblAccount WHERE Email = ?"
Set statement = ##class(%SQL.Statement).%New()
Set status = statement.%Prepare(sql)
Set resultSet = statement.%Execute("user@example.com")

// Check if found
If resultSet.%Next() {
  Set accountId = resultSet.%Get("ID")

  // Open the object
  Set account = ##class(SEM.tblAccount).%OpenId(accountId)
  Write "Password: ", account.Password
}
```

Update account:

```objectscript
// Open existing account
Set account = ##class(SEM.tblAccount).%OpenId(5)

// Modify property
Set account.Name = "佐藤花子"

// Save changes
Do account.%Save()
```

Delete account:

```objectscript
```

```
// Method 1: Delete by ID
Do ##class(SEM.tblAccount).%DeleteId(5)

// Method 2: Open then delete
Set account = ##class(SEM.tblAccount).%OpenId(5)
Do account.%Delete()
```

---

18. SEM.tblEmployee.cls

Purpose: Employee information database class। সব employee data store করে।

```objectscript
```

```
/// Employee information table
Class SEM.tblEmployee Extends %Persistent
{

/// Employee ID (5 digits, unique)
Property EmployeeId As %String(MAXLEN = 5) [ Required ];

/// Full name
Property Name As %String(MAXLEN = 64) [ Required ];

/// Kana name (Katakana)
Property KanaName As %String(MAXLEN = 64);

/// Sex (1=Male, 2=Female)
Property Sex As %Integer [ Required ];

/// Postal code (e.g., 100-0001)
Property PostCode As %String(MAXLEN = 8);

/// Full address
Property Address As %String(MAXLEN = 1024);

/// Phone number
Property PhoneNumber As %String(MAXLEN = 13);

/// Department name
Property Department As %String(MAXLEN = 64);

/// Retirement flag (0=Active, 1=Retired)
Property RetireFlg As %Boolean [ Required ];

/// Soft delete flag (0=Active, 1=Deleted)
Property deleteFlg As %Boolean [ Required ];

/// Last update timestamp
Property upDateTime As %DateTime [ Required ];

/// Ensure EmployeeId is unique among non-deleted records
Index EmployeeIdIndex On (EmployeeId, deleteFlg) [ Unique ];

}
```

Line-by-Line Explanation:

Line 1-3: Class Definition

```objectscript
/// Employee information table
Class SEM.tblEmployee Extends %Persistent
{
```

- Same structure as tblAccount
- Extends %Persistent for database persistence

Line 5-6: EmployeeId Property

```objectscript
/// Employee ID (5 digits, unique)
Property EmployeeId As %String(MAXLEN = 5) [ Required ];
```

Why String for ID?

```objectscript
// String allows:
"00123"  → Leading zeros preserved
"12345"  → Regular number
"ABC12"  → Alphanumeric (if needed)

// Integer would lose leading zeros:
00123 → stored as 123 (wrong!)
```

Why MAXLEN = 5?

- Business rule: Employee IDs are exactly 5 digits
- Examples: "12345", "00001", "99999"
- Database validates length

Line 8-9: Name Property

```objectscript
/// Full name
Property Name As %String(MAXLEN = 64) [ Required ];
```

- Required field (can't be empty)
- 64 characters sufficient for Japanese names
- UTF-8 encoding supports kanji

Japanese Name Example:

```objectscript
// Full kanji name fits easily:
"山田太郎" → 4 characters
"田中花子" → 4 characters
"佐藤一郎太郎" → 6 characters

// Even long names:
"長谷川美智子" → 6 characters
// Well under 64 char limit
```

Line 11-12: KanaName Property

```objectscript
/// Kana name (Katakana)
Property KanaName As %String(MAXLEN = 64);
```

- **Optional** (no Required keyword)
- Katakana pronunciation of name
- Examples: "ヤマダタロウ", "タナカハナコ"

Why Kana Name?

```objectscript
```

```
// Japanese names can be read multiple ways:
Name: "山田"
KanaName: "ヤマダ" (Yamada)
// OR
KanaName: "サンダ" (Sanda)

// Kana clarifies pronunciation
// Useful for:
// - Sorting (phonetic order)
// - Phone directory
// - Furigana display
```

Line 14-15: Sex Property

```objectscript
/// Sex (1=Male, 2=Female)
Property Sex As %Integer [ Required ];
```

- **Integer type**: More efficient than string
- **1 = Male (男性)**
- **2 = Female (女性)**
- Required field

Why Integer?

```objectscript
```

```
// ✅ Integer (efficient):
Sex = 1  → 4 bytes
Sex = 2  → 4 bytes

// ❌ String (wasteful):
Sex = "Male"    → 4 bytes
Sex = "Female"  → 6 bytes
Sex = "男性"    → 6 bytes (2 chars × 3 bytes UTF-8)

// Integer also:
// - Faster comparisons
// - Easier validation
// - Language-independent
```

Line 17-18: PostCode Property

```objectscript
/// Postal code (e.g., 100-0001)
Property PostCode As %String(MAXLEN = 8);
```

- Optional field
- Japanese postal code format: "XXX-XXXX" (8 characters with hyphen)
- Examples: "100-0001", "150-0002"

Japanese Postal Code:

```objectscript
```

```
// Format: 3 digits - hyphen - 4 digits
"100-0001" → Tokyo Chiyoda-ku
"150-0002" → Tokyo Shibuya-ku
"060-0001" → Hokkaido Sapporo

// Length = 8 (including hyphen)
```

Line 20-21: Address Property

objectscript

```
/// Full address
Property Address As %String(MAXLEN = 1024);
```

- **MAXLEN = 1024**: Much longer than other fields
- Japanese addresses can be quite long

Why 1024 characters?

objectscript

```
// Japanese address example:
"東京都千代田区千代田1-1 千代田ビル3階301号室"
// Translation: Tokyo, Chiyoda-ku, Chiyoda 1-1, Chiyoda Building 3F Room 301

// Full addresses can include:
// - Prefecture (都道府県)
// - City (市区町村)
// - Town (町丁目)
// - Block/Building (番地・ビル名)
// - Floor/Room (階・部屋番号)

// 1024 chars ensures no truncation
```

Line 23-24: PhoneNumber Property

```objectscript
/// Phone number
Property PhoneNumber As %String(MAXLEN = 13);
```

- Optional field
- 13 characters for Japanese mobile format

Japanese Phone Format:

```objectscript
```

```
// Mobile: 090-1234-5678 (13 chars)
// Mobile: 080-9876-5432 (13 chars)
// Mobile: 070-1111-2222 (13 chars)

// Landline: 03-1234-5678 (12 chars)
// Landline: 06-9876-5432 (12 chars)

// Format: XXX-XXXX-XXXX (mobile)
// Format: XX-XXXX-XXXX (landline)
```

Line 26-27: Department Property

```objectscript
/// Department name
Property Department As %String(MAXLEN = 64);
```

- Optional field
- Department/Division name
- Examples: "営業部", "技術部", "総務部"

Line 29-30: RetireFlg Property

```objectscript
/// Retirement flag (0=Active, 1=Retired)
Property RetireFlg As %Boolean [ Required ];
```

- **%Boolean type**: True or False

- **True (1)**: Employee retired
- **False (0)**: Employee active
- Required field (must be set)

Boolean in ObjectScript:

```objectscript
// Setting boolean:
Set employee.RetireFlg = 1  → True (retired)
Set employee.RetireFlg = 0  → False (active)

// Checking boolean:
If employee.RetireFlg {
  Write "Retired"
} Else {
  Write "Active"
}

// In database:
// Stored as BIT (0 or 1)
```

Line 32-33: deleteFlg Property

```objectscript
/// Soft delete flag (0=Active, 1=Deleted)
Property deleteFlg As %Boolean [ Required ];
```

- **Soft delete pattern**: Don't actually delete records

- **True (1)**: Record deleted (hidden)
- **False (0)**: Record active (visible)

Soft Delete Explained:

```objectscript
// Hard Delete (BAD):
Do ##class(SEM.tblEmployee).%DeleteId(5)
// Record permanently removed from database
// Cannot recover!
// Breaks referential integrity

// Soft Delete (GOOD):
Set employee = ##class(SEM.tblEmployee).%OpenId(5)
Set employee.deleteFlg = 1
Do employee.%Save()
// Record still in database
// Just marked as deleted
// Can recover if needed!

// Query active records:
SELECT * FROM SEM.tblEmployee WHERE deleteFlg = 0
```

Why Soft Delete?

1. **Data Recovery**: Can undelete if mistake
2. **Audit Trail**: Keep history of all employees
3. **Referential Integrity**: Related records don't break
4. **Compliance**: Legal requirements to keep records

5. **Analytics**: Historical data for reports

Line 35-36: upDateTime Property

```objectscript
/// Last update timestamp
Property upDateTime As %DateTime [ Required ];
```

- **%DateTime type**: Date + Time
- Tracks when record was last modified
- Required field (always set on save)

DateTime Format:

```objectscript
// IRIS DateTime format:
// "YYYY-MM-DD HH:MM:SS"

// Examples:
"2024-12-25 14:30:45"
"2025-01-01 00:00:00"
"2024-06-15 09:15:30"

// Getting current datetime:
Set currentDateTime = $ZDATETIME($HOROLOG, 3)
// $HOROLOG = internal date/time format
// 3 = format code (YYYY-MM-DD HH:MM:SS)
```

Usage Pattern:

```objectscript
// On create:
Set employee.upDateTime = $ZDATETIME($HOROLOG, 3)

// On update:
Set employee = ##class(SEM.tblEmployee).%OpenId(5)
Set employee.Name = "New Name"
Set employee.upDateTime = $ZDATETIME($HOROLOG, 3)
Do employee.%Save()
```

Line 38-39: Composite Unique Index

```objectscript
/// Ensure EmployeeId is unique among non-deleted records
Index EmployeeIdIndex On (EmployeeId, deleteFlg) [ Unique ];
```

This is COMPLEX and IMPORTANT!

Composite Index: Index on multiple columns

```objectscript
// Index on TWO columns:
// 1. EmployeeId
// 2. deleteFlg

// Why?
// Allow same EmployeeId if one is deleted
```

Example Scenario:

```objectscript
// Employee created:
EmployeeId: "12345", deleteFlg: 0 (active)  → OK

// Try to create duplicate:
EmployeeId: "12345", deleteFlg: 0 (active)  → ERROR! Duplicate

// Delete first employee:
EmployeeId: "12345", deleteFlg: 1 (deleted)

// Now can reuse ID:
EmployeeId: "12345", deleteFlg: 0 (active)  → OK!

// Database sees these as different:
// ("12345", 0) ≠ ("12345", 1)
```

Without composite index:

```objectscript
```

```
// Simple index on EmployeeId only:
Index EmployeeIdIndex On EmployeeId [ Unique ];

// Problem:
EmployeeId: "12345", deleteFlg: 0  → OK
// Delete this employee (set deleteFlg = 1)
EmployeeId: "12345", deleteFlg: 1  → Still in database

// Try to create new employee:
EmployeeId: "12345", deleteFlg: 0  → ERROR! ID exists
// Cannot reuse employee IDs ever!
```

With composite index:

```objectscript
// Index on (EmployeeId, deleteFlg):
Index EmployeeIdIndex On (EmployeeId, deleteFlg) [ Unique ];

// Uniqueness check:
("12345", 0) → OK
("12345", 1) → OK (different from above)
("12345", 0) → ERROR! Duplicate of first

// Can have:
// - One active employee with ID "12345"
// - Multiple deleted employees with ID "12345"
```

Generated SQL:

```sql
```

```sql
CREATE UNIQUE INDEX EmployeeIdIndex
ON SEM.tblEmployee (EmployeeId, deleteFlg);

-- Ensures:
-- 1. Fast lookup by EmployeeId
-- 2. Only ONE active employee per ID
-- 3. Multiple deleted employees OK
```

Line 41: Closing Brace

```objectscript
}
```

- Class definition ends

Generated Database Table:

```sql
```

```
-- Automatically generated by IRIS:
CREATE TABLE SEM.tblEmployee (
  ID INTEGER PRIMARY KEY AUTO_INCREMENT,
  EmployeeId VARCHAR(5) NOT NULL,
  Name VARCHAR(64) NOT NULL,
  KanaName VARCHAR(64),
  Sex INTEGER NOT NULL,
  PostCode VARCHAR(8),
  Address VARCHAR(1024),
  PhoneNumber VARCHAR(13),
  Department VARCHAR(64),
  RetireFlg BIT NOT NULL,
  deleteFlg BIT NOT NULL,
  upDateTime DATETIME NOT NULL
);

CREATE UNIQUE INDEX EmployeeIdIndex
ON SEM.tblEmployee (EmployeeId, deleteFlg);
```

Complete CRUD Examples:

1. Create Employee:

```objectscript
```

```
// Create new object
Set emp = ##class(SEM.tblEmployee).%New()

// Set required properties
Set emp.EmployeeId = "12345"
Set emp.Name = "山田太郎"
Set emp.Sex = 1
Set emp.RetireFlg = 0
Set emp.deleteFlg = 0
Set emp.upDateTime = $ZDATETIME($HOROLOG, 3)

// Set optional properties
Set emp.KanaName = "ヤマダタロウ"
Set emp.PostCode = "100-0001"
Set emp.Address = "東京都千代田区千代田1-1"
Set emp.PhoneNumber = "090-1234-5678"
Set emp.Department = "営業部"

// Save to database
Set status = emp.%Save()

If $$$ISOK(status) {
  Write "Employee created with ID: ", emp.%Id()
} Else {
  Write "Error: ", $System.Status.GetErrorText(status)
}
```

2. Read Employee:

```
objectscript
```

```
// Get by database ID:
Set emp = ##class(SEM.tblEmployee).%OpenId(5)
Write "Name: ", emp.Name

// Get by EmployeeId using SQL:
Set sql = "SELECT ID FROM SEM.tblEmployee WHERE EmployeeId = ? AND deleteFlg = 0"
Set statement = ##class(%SQL.Statement).%New()
Do statement.%Prepare(sql)
Set resultSet = statement.%Execute("12345")

If resultSet.%Next() {
  Set empId = resultSet.%Get("ID")
  Set emp = ##class(SEM.tblEmployee).%OpenId(empId)
  Write "Name: ", emp.Name
}
```

3. Update Employee:

```objectscript
// Open existing employee
Set emp = ##class(SEM.tblEmployee).%OpenId(5)

// Modify properties
Set emp.Name = "佐藤花子"
Set emp.Department = "技術部"
Set emp.RetireFlg = 1
Set emp.upDateTime = $ZDATETIME($HOROLOG, 3)

// Save changes
Do emp.%Save()
```

4. Delete Employee (Soft):

```objectscript
// Open employee
Set emp = ##class(SEM.tblEmployee).%OpenId(5)

// Set delete flag
Set emp.deleteFlg = 1
Set emp.upDateTime = $ZDATETIME($HOROLOG, 3)

// Save
Do emp.%Save()

// Employee still in database but hidden
```

5. Query Active Employees:

```objectscript
```

```
// SQL query for active employees
Set sql = "SELECT ID, EmployeeId, Name, Sex, RetireFlg "_
          "FROM SEM.tblEmployee "_
          "WHERE deleteFlg = 0 "_
          "ORDER BY upDateTime DESC"

Set statement = ##class(%SQL.Statement).%New()
Do statement.%Prepare(sql)
Set resultSet = statement.%Execute()

// Loop through results
While resultSet.%Next() {
  Set id = resultSet.%Get("ID")
  Set empId = resultSet.%Get("EmployeeId")
  Set name = resultSet.%Get("Name")

  Write "ID: ", id, " | EmployeeId: ", empId, " | Name: ", name, !
}
```

These two database classes form the foundation of the entire application. Next, I'll explain the REST API class that connects frontend to these database classes!

Should I continue with Part 7 - REST API Handler? That's where all the magic happens - HTTP requests, JSON parsing, database operations, and responses!