# 💻 Complete Code Explanation - Part 8

## Backend (IRIS ObjectScript) - REST API CRUD Methods (Part 2)

---

### Continuing from Part 7...

Now let's cover the remaining CRUD operations which are more complex!

---

### Method 4: Get Employee By ID

```objectscript
```

```
/// Get single employee by database ID
ClassMethod GetEmployeeById(id As %String) As %Status
{
  Set result = {}
  Set result.data = {}

  Try {
    // Validate ID parameter
    If (id = "") {
      Set result.message = "Employee ID is required."
      Do ..WriteJSONResponse(result)
      Return $$$OK
    }

    // Open employee object
    Set employee = ##class(SEM.tblEmployee).%OpenId(id)

    // Check if employee exists
    If '$IsObject(employee) {
      Set result.message = "Employee not found."
      Do ..WriteJSONResponse(result)
      Return $$$OK
    }

    // Check if deleted
    If (employee.deleteFlg = 1) {
      Set result.message = "Employee not found."
      Do ..WriteJSONResponse(result)
      Return $$$OK
    }

    // Build response object
```

```
    Set empData = {}
    Set empData.id = employee.%Id()
    Set empData.EmployeeId = employee.EmployeeId
    Set empData.Name = $ZCONVERT(employee.Name, "O", "UTF8")
    Set empData.KanaName = $ZCONVERT(employee.KanaName, "O", "UTF8")
    Set empData.Sex = employee.Sex
    Set empData.PostCode = employee.PostCode
    Set empData.Address = $ZCONVERT(employee.Address, "O", "UTF8")
    Set empData.PhoneNumber = employee.PhoneNumber
    Set empData.Department = $ZCONVERT(employee.Department, "O", "UTF8")
    Set empData.RetireFlg = employee.RetireFlg
    Set empData.upDateTime = employee.upDateTime

    Set result.data = empData
    Set result.message = "Success"

  } Catch ex {
    Set result.message = "Error: "_ex.DisplayString()
  }

  Do ..WriteJSONResponse(result)
  Return $$$OK
}
```

**Line-by-Line Explanation:**

**Line 2: Method Signature with Parameter**

```
objectscript

ClassMethod GetEmployeeById(id As %String) As %Status
```

- **id As %String**: Method parameter
- **Type declaration**: Ensures type safety
- This `id` comes from URL parameter

**How URL parameters work:**

```xml
<!-- Route definition: -->
<Route Url="/employee/:id" Method="GET" Call="GetEmployeeById"/>

<!-- Example request: -->
GET /employee/42

<!-- IRIS extracts: -->
id = "42"

<!-- Calls: -->
GetEmployeeById("42")
```

**Line 4-5: Initialize Response**

```objectscript
Set result = {}
Set result.data = {}
```

- **result.data**: Nested object for employee data
- Structure:

```json
{
  "message": "...",
  "data": {
    "id": 1,
    "Name": "...",
    ...
  }
}
```

**Line 8-13: Validate ID Parameter**

```objectscript
If (id = "") {
  Set result.message = "Employee ID is required."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}
```

- Check if id parameter provided
- Early return if missing

**Line 15-16: Open Employee Object**

```objectscript
Set employee = ##class(SEM.tblEmployee).%OpenId(id)
```

**%OpenId() Explained:**

- Opens persistent object by database ID
- Returns object instance if found
- Returns "" (empty string) if not found

**Example:**

```
objectscript

// Database has employee with ID = 5
Set emp = ##class(SEM.tblEmployee).%OpenId(5)
// emp is now an object with all properties

Write emp.Name          // "山田太郎"
Write emp.EmployeeId    // "12345"

// Database does NOT have ID = 999
Set emp = ##class(SEM.tblEmployee).%OpenId(999)
// emp = "" (empty string, not an object)
```

**Line 18-23: Check if Object Exists**

```
objectscript

If '$IsObject(employee) {
  Set result.message = "Employee not found."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}
```

**$IsObject() function:**

- Returns 1 (true) if variable is an object
- Returns 0 (false) if not an object

**Why ' (apostrophe)?**

```objectscript
// ' = NOT operator in ObjectScript

If '$IsObject(employee) {
  // If NOT an object
}

// Equivalent to:
If $IsObject(employee) = 0 {
  // If not an object
}

// Similar patterns:
If 'condition        // If NOT condition
If '(x = 5)          // If NOT (x equals 5)
```

**Comparison with other languages:**

```javascript
```

```
// JavaScript:
if (!isObject(employee)) {
  // Not an object
}

// Python:
if not is_object(employee):
    # Not an object

// ObjectScript:
If '$IsObject(employee) {
  // Not an object
}
```

**Line 25-29: Check Soft Delete Flag**

```
objectscript

If (employee.deleteFlg = 1) {
  Set result.message = "Employee not found."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}
```

- Even if employee exists in database
- If deleted (deleteFlg = 1) → treat as "not found"
- Don't expose deleted records to frontend

**Why check deleteFlg here?**

```objectscript
// Scenario:
// 1. Employee ID 42 exists but is soft-deleted
// 2. Frontend tries: GET /employee/42
// 3. Without check: Returns deleted employee data
// 4. With check: Returns "Employee not found"

// This maintains soft delete illusion
// Deleted records appear to not exist
```

### Line 31-43: Build Response Object

```objectscript
Set empData = {}
Set empData.id = employee.%Id()
Set empData.EmployeeId = employee.EmployeeId
Set empData.Name = $ZCONVERT(employee.Name, "O", "UTF8")
// ... all other properties
```

### Why create new object instead of returning employee directly?

```objectscript
```

```
// ❌ Can't do this:
Set result.data = employee
// employee is persistent object, not dynamic object

// ✅ Must do this:
Set empData = {}  // Create dynamic object
Set empData.Name = employee.Name  // Copy properties
Set result.data = empData  // Assign to result
```

**Accessing object properties:**

```
objectscript

// Read property:
Set name = employee.Name

// Write property:
Set employee.Name = "New Name"

// Get database ID:
Set id = employee.%Id()
```

**Line 36-42: UTF-8 Conversion for Output**

```
objectscript

Set empData.Name = $ZCONVERT(employee.Name, "O", "UTF8")
Set empData.KanaName = $ZCONVERT(employee.KanaName, "O", "UTF8")
Set empData.Address = $ZCONVERT(employee.Address, "O", "UTF8")
Set empData.Department = $ZCONVERT(employee.Department, "O", "UTF8")
```

- **"O" direction**: Output (IRIS → UTF-8)
- Ensures Japanese characters sent correctly
- Only needed for text fields that may contain Japanese

**Which fields need conversion?**

```
objectscript

// ✅ Need conversion (may contain Japanese):
Name       → "山田太郎"
KanaName   → "ヤマダタロウ"
Address    → "東京都..."
Department → "営業部"

// ❌ Don't need conversion (numbers/codes):
id         → 5
EmployeeId → "12345"
Sex        → 1
PostCode   → "100-0001"
PhoneNumber → "090-1234-5678"
RetireFlg  → 0
```

**Line 45-46: Set Success Response**

```
objectscript

Set result.data = empData
Set result.message = "Success"
```

**Final response structure:**

```json
{
  "message": "Success",
  "data": {
    "id": 5,
    "EmployeeId": "12345",
    "Name": "山田太郎",
    "KanaName": "ヤマダタロウ",
    "Sex": 1,
    "PostCode": "100-0001",
    "Address": "東京都千代田区...",
    "PhoneNumber": "090-1234-5678",
    "Department": "営業部",
    "RetireFlg": false,
    "upDateTime": "2024-12-25 14:30:00"
  }
}
```

---

## Method 5: Create Employee

```objectscript
```

```
/// Create new employee
ClassMethod CreateEmployee() As %Status
{
  Set result = {}
  Set result.message = ""

  Try {
    // Parse request body
    Set requestObject = ##Class(%DynamicAbstractObject).%FromJSON(%request.Content)

    // Extract data
    Set reqEmployeeId = requestObject.employeeId
    Set reqName = requestObject.name
    Set reqKanaName = requestObject.kanaName
    Set reqSex = requestObject.sex
    Set reqPostCode = requestObject.postCode
    Set reqAddress = requestObject.address
    Set reqPhoneNumber = requestObject.phoneNumber
    Set reqDepartment = requestObject.department
    Set reqRetireFlg = requestObject.retireFlg

    // Validation: Required fields
    If (reqEmployeeId = "") {
      Set result.message = "Employee ID is required."
      Do ..WriteJSONResponse(result)
      Return $$$OK
    }

    If ($Length(reqEmployeeId) '= 5) {
      Set result.message = "Employee ID must be exactly 5 digits."
      Do ..WriteJSONResponse(result)
      Return $$$OK
```

```
}

If (reqName = "") {
  Set result.message = "Name is required."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}

If (reqSex = "") {
  Set result.message = "Sex is required."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}

// Check for duplicate EmployeeId (active records only)
Set sqlQuery = "SELECT ID FROM SEM.tblEmployee "_
            "WHERE EmployeeId = ? AND deleteFlg = 0"
Set statement = ##class(%SQL.Statement).%New()
Do statement.%Prepare(sqlQuery)
Set resultSet = statement.%Execute(reqEmployeeId)

If resultSet.%Next() {
  Set result.message = "Employee ID already exists."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}

// Create new employee object
Set newEmployee = ##class(SEM.tblEmployee).%New()

// Set properties
Set newEmployee.EmployeeId = reqEmployeeId
```

```
    Set newEmployee.Name = $ZCONVERT(reqName, "I", "UTF8")
    Set newEmployee.KanaName = $ZCONVERT(reqKanaName, "I", "UTF8")
    Set newEmployee.Sex = reqSex
    Set newEmployee.PostCode = reqPostCode
    Set newEmployee.Address = $ZCONVERT(reqAddress, "I", "UTF8")
    Set newEmployee.PhoneNumber = reqPhoneNumber
    Set newEmployee.Department = $ZCONVERT(reqDepartment, "I", "UTF8")
    Set newEmployee.RetireFlg = reqRetireFlg
    Set newEmployee.deleteFlg = 0  // Active (not deleted)
    Set newEmployee.upDateTime = $ZDATETIME($HOROLOG, 3)

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

    If $$$ISOK(status) {
      Set result.message = "Employee created successfully."
      Set result.id = newEmployee.%Id()
    } Else {
      Set result.message = "Failed to create employee."
      Set result.error = $System.Status.GetErrorText(status)
    }

  } Catch ex {
    Set result.message = "Error: "_ex.DisplayString()
  }

  Do ..WriteJSONResponse(result)
  Return $$$OK
}
```

**Line-by-Line Explanation:**

**Line 9-19: Extract All Fields from JSON**

```objectscript
Set reqEmployeeId = requestObject.employeeId
Set reqName = requestObject.name
Set reqKanaName = requestObject.kanaName
// ... etc
```

**Request body example:**

```json
{
  "employeeId": "12345",
  "name": "山田太郎",
  "kanaName": "ヤマダタロウ",
  "sex": 1,
  "postCode": "100-0001",
  "address": "東京都千代田区...",
  "phoneNumber": "090-1234-5678",
  "department": "営業部",
  "retireFlg": false
}
```

**Naming convention:**

```objectscript
```

```
// Frontend (JavaScript/JSON): camelCase
{
  "employeeId": "12345",
  "kanaName": "...",
}

// Backend (ObjectScript): PascalCase
Property EmployeeId
Property KanaName

// We transform between them:
Set reqEmployeeId = requestObject.employeeId
Set newEmployee.EmployeeId = reqEmployeeId
```

### Line 22-27: Validate Employee ID Presence

```
objectscript

If (reqEmployeeId = "") {
  Set result.message = "Employee ID is required."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}
```

- Check if employeeId provided
- Empty string check

### Line 29-33: Validate Employee ID Length

```
objectscript
```

```
If ($Length(reqEmployeeId) '= 5) {
  Set result.message = "Employee ID must be exactly 5 digits."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}
```

### $Length() function:

objectscript

```
// Returns length of string
$Length("Hello")      → 5
$Length("12345")      → 5
$Length("123")        → 3
$Length("")           → 0

// '= means "not equal"
If ($Length(text) '= 5) {
  // If length is NOT 5
}
```

### Why '= instead of != ?

objectscript

```
// ObjectScript uses:
=   → equal
'=  → not equal
<   → less than
>   → greater than
<=  → less than or equal
'<  → not less than (same as >=)

// NOT like JavaScript:
== or ===
!=
```

**Line 35-43: Validate Other Required Fields**

```
objectscript

If (reqName = "") {
  Set result.message = "Name is required."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}

If (reqSex = "") {
  Set result.message = "Sex is required."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}
```

- Name is required
- Sex is required

- Other fields optional (no validation)

### Line 46-58: Check Duplicate Employee ID

```
objectscript

Set sqlQuery = "SELECT ID FROM SEM.tblEmployee "_
            "WHERE EmployeeId = ? AND deleteFlg = 0"
Set statement = ##class(%SQL.Statement).%New()
Do statement.%Prepare(sqlQuery)
Set resultSet = statement.%Execute(reqEmployeeId)

If resultSet.%Next() {
  Set result.message = "Employee ID already exists."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}
```

### Why check deleteFlg = 0?

```
objectscript
```

```
// Without deleteFlg check:
SELECT ID FROM SEM.tblEmployee WHERE EmployeeId = ?
// Finds BOTH active and deleted employees
// Cannot reuse deleted employee IDs

// With deleteFlg check:
SELECT ID FROM SEM.tblEmployee WHERE EmployeeId = ? AND deleteFlg = 0
// Finds ONLY active employees
// CAN reuse deleted employee IDs ✅

// Example:
// Employee "12345" was deleted (deleteFlg = 1)
// New employee can use "12345" again
// Because query only checks active (deleteFlg = 0)
```

### Line 60-61: Create New Object

```objectscript
Set newEmployee = ##class(SEM.tblEmployee).%New()
```

- Creates new instance
- Not yet saved to database
- All properties initially empty/default

### Line 63-75: Set All Properties

```objectscript
```

```
Set newEmployee.EmployeeId = reqEmployeeId
Set newEmployee.Name = $ZCONVERT(reqName, "I", "UTF8")
Set newEmployee.KanaName = $ZCONVERT(reqKanaName, "I", "UTF8")
Set newEmployee.Sex = reqSex
Set newEmployee.PostCode = reqPostCode
Set newEmployee.Address = $ZCONVERT(reqAddress, "I", "UTF8")
Set newEmployee.PhoneNumber = reqPhoneNumber
Set newEmployee.Department = $ZCONVERT(reqDepartment, "I", "UTF8")
Set newEmployee.RetireFlg = reqRetireFlg
Set newEmployee.deleteFlg = 0
Set newEmployee.upDateTime = $ZDATETIME($HOROLOG, 3)
```

### UTF-8 Conversion "I" (Input):

```
objectscript

// Japanese text from HTTP request (UTF-8)
reqName = "山田太郎"

// Convert to IRIS internal format
Set newEmployee.Name = $ZCONVERT(reqName, "I", "UTF8")

// Without conversion, might get mojibake!
```

### Setting timestamps:

```
objectscript
```

```
Set newEmployee.upDateTime = $ZDATETIME($HOROLOG, 3)
//                          ↓       ↓       ↓
//                       Function  Internal  Format
//                                 datetime   code

// $HOROLOG = IRIS internal date/time
// Example: 66777,52345

// $ZDATETIME(horolog, 3) = Convert to string
// Format 3 = "YYYY-MM-DD HH:MM:SS"
// Example: "2024-12-25 14:32:15"
```

**$HOROLOG Explained:**

```
objectscript

// $HOROLOG returns two integers separated by comma
// Example: 66777,52345

// First number: Days since December 31, 1840
// 66777 = Number of days since Dec 31, 1840

// Second number: Seconds since midnight
// 52345 = 14:32:25 (14 hours, 32 minutes, 25 seconds)

// Convert to readable format:
$ZDATETIME($HOROLOG, 3) → "2024-12-25 14:32:25"
```

**Format codes for $ZDATETIME:**

```
objectscript
```

```
$ZDATETIME($HOROLOG, 1) → "12/25/2024"
$ZDATETIME($HOROLOG, 2) → "25 Dec 2024"
$ZDATETIME($HOROLOG, 3) → "2024-12-25 14:32:25"
$ZDATETIME($HOROLOG, 4) → "25/12/2024 14:32:25"
```

### Line 74-75: Initialize System Fields

```
objectscript

Set newEmployee.deleteFlg = 0  // Active
Set newEmployee.upDateTime = $ZDATETIME($HOROLOG, 3)  // Current timestamp
```

- **deleteFlg = 0**: New employee is active (not deleted)
- **upDateTime**: Creation timestamp

### Line 77-86: Save and Handle Result

```
objectscript

Set status = newEmployee.%Save()

If $$$ISOK(status) {
  Set result.message = "Employee created successfully."
  Set result.id = newEmployee.%Id()
} Else {
  Set result.message = "Failed to create employee."
  Set result.error = $System.Status.GetErrorText(status)
}
```

**%Save() operation:**

objectscript

```
// Before save:
newEmployee.%Id() → ""  (no ID yet)

// Call save:
Set status = newEmployee.%Save()

// After successful save:
newEmployee.%Id() → "5"  (database assigned ID)

// The object is now persisted to database
// Can be opened later with %OpenId(5)
```

**Error handling:**

objectscript

```
If $$$ISOK(status) {
  // Success path
} Else {
  // Error path
  Set errorMsg = $System.Status.GetErrorText(status)
  // Example errors:
  // - "Unique constraint violation"
  // - "Required field missing"
  // - "Database connection failed"
}
```

---

## Method 6: Update Employee

objectscript

```
/// Update existing employee
ClassMethod UpdateEmployee(id As %String) As %Status
{
  Set result = {}
  Set result.message = ""

  Try {
    // Validate ID
    If (id = "") {
      Set result.message = "Employee ID is required."
      Do ..WriteJSONResponse(result)
      Return $$$OK
    }

    // Open existing employee
    Set employee = ##class(SEM.tblEmployee).%OpenId(id)

    If '$IsObject(employee) {
      Set result.message = "Employee not found."
      Do ..WriteJSONResponse(result)
      Return $$$OK
    }

    If (employee.deleteFlg = 1) {
      Set result.message = "Employee not found."
      Do ..WriteJSONResponse(result)
      Return $$$OK
    }

    // Parse request body
    Set requestObject = ##Class(%DynamicAbstractObject).%FromJSON(%request.Content)
```

```
// Extract data
Set reqName = requestObject.name
Set reqKanaName = requestObject.kanaName
Set reqSex = requestObject.sex
Set reqPostCode = requestObject.postCode
Set reqAddress = requestObject.address
Set reqPhoneNumber = requestObject.phoneNumber
Set reqDepartment = requestObject.department
Set reqRetireFlg = requestObject.retireFlg

// Validation: Required fields
If (reqName = "") {
  Set result.message = "Name is required."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}

If (reqSex = "") {
  Set result.message = "Sex is required."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}

// Update properties (NOT EmployeeId - it's immutable)
Set employee.Name = $ZCONVERT(reqName, "I", "UTF8")
Set employee.KanaName = $ZCONVERT(reqKanaName, "I", "UTF8")
Set employee.Sex = reqSex
Set employee.PostCode = reqPostCode
Set employee.Address = $ZCONVERT(reqAddress, "I", "UTF8")
Set employee.PhoneNumber = reqPhoneNumber
Set employee.Department = $ZCONVERT(reqDepartment, "I", "UTF8")
Set employee.RetireFlg = reqRetireFlg
```

```
    Set employee.upDateTime = $ZDATETIME($HOROLOG, 3)

    // Save changes
    Set status = employee.%Save()

    If $$$ISOK(status) {
      Set result.message = "Employee updated successfully."
    } Else {
      Set result.message = "Failed to update employee."
      Set result.error = $System.Status.GetErrorText(status)
    }

  } Catch ex {
    Set result.message = "Error: "_ex.DisplayString()
  }

  Do ..WriteJSONResponse(result)
  Return $$$OK
}
```

**Line-by-Line Explanation:**

**Key Difference from Create:**

- **Opens existing** object instead of creating new
- **Updates** properties instead of setting them initially
- **Does NOT change** EmployeeId (immutable)

**Line 2: Method with URL Parameter**

```
objectscript
```

```
ClassMethod UpdateEmployee(id As %String) As %Status
```

### URL routing:

```xml
<Route Url="/employee/:id" Method="PUT" Call="UpdateEmployee"/>

<!-- Example: -->
PUT /employee/42
→ UpdateEmployee("42")
```

### Line 15-16: Open Existing Employee

```objectscript
Set employee = ##class(SEM.tblEmployee).%OpenId(id)
```

### %OpenId vs %New:

```objectscript
```

```
// Create (POST):
Set emp = ##class(SEM.tblEmployee).%New()
// Creates NEW object
// No database ID yet
// %Id() = ""

// Update (PUT):
Set emp = ##class(SEM.tblEmployee).%OpenId(5)
// Opens EXISTING object from database
// Has database ID
// %Id() = "5"
```

**Line 17-27: Validate Employee Exists**

```
objectscript

If '$IsObject(employee) {
  Set result.message = "Employee not found."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}

If (employee.deleteFlg = 1) {
  Set result.message = "Employee not found."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}
```

- Check if ID exists
- Check if not deleted

- Same validation as GetEmployeeById

**Line 29-40: Parse Request Body**

```objectscript
Set requestObject = ##Class(%DynamicAbstractObject).%FromJSON(%request.Content)

Set reqName = requestObject.name
Set reqKanaName = requestObject.kanaName
// ... etc
```

**Request body (PUT):**

```json
{
  "name": "佐藤花子",
  "kanaName": "サトウハナコ",
  "sex": 2,
  "postCode": "150-0001",
  "address": "東京都渋谷区...",
  "phoneNumber": "080-9876-5432",
  "department": "技術部",
  "retireFlg": true
}
```

**Notice: No employeeId in request!**

- EmployeeId cannot be changed (business rule)
- Frontend disables the field in edit mode

- Backend ignores it even if sent

**Line 42-54: Validation**

```objectscript
If (reqName = "") {
  Set result.message = "Name is required."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}

If (reqSex = "") {
  Set result.message = "Sex is required."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}
```

- Same validation as Create
- Required fields must not be empty

**Line 56-65: Update Properties**

```objectscript
```

```
Set employee.Name = $ZCONVERT(reqName, "I", "UTF8")
Set employee.KanaName = $ZCONVERT(reqKanaName, "I", "UTF8")
Set employee.Sex = reqSex
Set employee.PostCode = reqPostCode
Set employee.Address = $ZCONVERT(reqAddress, "I", "UTF8")
Set employee.PhoneNumber = reqPhoneNumber
Set employee.Department = $ZCONVERT(reqDepartment, "I", "UTF8")
Set employee.RetireFlg = reqRetireFlg
Set employee.upDateTime = $ZDATETIME($HOROLOG, 3)
```

**Important notes:**

```
objectscript

// ❌ NOT updated (immutable):
// employee.EmployeeId = ...
// employee.deleteFlg = ...  (only changed on delete)

// ✅ Updated:
// All other fields
// upDateTime always updated to current time
```

**Update vs Initial Set:**

```
objectscript
```

```
// Both use same syntax:
Set employee.Name = "New Name"

// But behavior differs:

// Create (NEW object):
Set emp = ##class(SEM.tblEmployee).%New()
Set emp.Name = "山田太郎"
// Setting property for first time

// Update (EXISTING object):
Set emp = ##class(SEM.tblEmployee).%OpenId(5)
Set emp.Name = "佐藤花子"
// Overwriting existing property value
```

### Line 67-76: Save Changes

```
objectscript

Set status = employee.%Save()

If $$$ISOK(status) {
  Set result.message = "Employee updated successfully."
} Else {
  Set result.message = "Failed to update employee."
  Set result.error = $System.Status.GetErrorText(status)
}
```

### %Save() on existing object:

```
objectscript
```

```
// Opens employee ID 5 from database:
Set emp = ##class(SEM.tblEmployee).%OpenId(5)
// Name = "山田太郎"
// Department = "営業部"

// Modify properties:
Set emp.Name = "佐藤花子"
Set emp.Department = "技術部"

// Save changes back to database:
Do emp.%Save()

// Database now updated:
// ID 5: Name = "佐藤花子", Department = "技術部"
```

---

## Method 7: Delete Employee (Soft Delete)

```
objectscript
```

```
/// Delete employee (soft delete)
ClassMethod DeleteEmployee(id As %String) As %Status
{
  Set result = {}
  Set result.message = ""

  Try {
    // Validate ID
    If (id = "") {
      Set result.message = "Employee ID is required."
      Do ..WriteJSONResponse(result)
      Return $$$OK
    }

    // Open existing employee
    Set employee = ##class(SEM.tblEmployee).%OpenId(id)

    If '$IsObject(employee) {
      Set result.message = "Employee not found."
      Do ..WriteJSONResponse(result)
      Return $$$OK
    }

    If (employee.deleteFlg = 1) {
      Set result.message = "Employee already deleted."
      Do ..WriteJSONResponse(result)
      Return $$$OK
    }

    // Soft delete: Set flag instead of actual deletion
    Set employee.deleteFlg = 1
    Set employee.upDateTime = $ZDATETIME($HOROLOG, 3)
```

```
    // Save changes
    Set status = employee.%Save()

    If $$$ISOK(status) {
      Set result.message = "Employee deleted successfully."
    } Else {
      Set result.message = "Failed to delete employee."
      Set result.error = $System.Status.GetErrorText(status)
    }

  } Catch ex {
    Set result.message = "Error: "_ex.DisplayString()
  }

  Do ..WriteJSONResponse(result)
  Return $$$OK
}
```

**Line-by-Line Explanation:**

**Line 2: Method Signature**

```
objectscript

ClassMethod DeleteEmployee(id As %String) As %Status
```

**URL routing:**

```
xml
```

```
<Route Url="/employee/:id" Method="DELETE" Call="DeleteEmployee"/>

<!-- Example: -->
DELETE /employee/42
→ DeleteEmployee("42")
```

### Line 15-16: Open Employee

```
objectscript

Set employee = ##class(SEM.tblEmployee).%OpenId(id)
```

- Must open object to modify it
- Can't delete without opening first

### Line 17-28: Validation

```
objectscript

If '$IsObject(employee) {
  Set result.message = "Employee not found."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}

If (employee.deleteFlg = 1) {
  Set result.message = "Employee already deleted."
  Do ..WriteJSONResponse(result)
  Return $$$OK
}
```

- Check if exists
- Check if already deleted (can't delete twice)

**Line 30-32: Soft Delete Implementation**

```
objectscript

Set employee.deleteFlg = 1
Set employee.upDateTime = $ZDATETIME($HOROLOG, 3)
```

**This is the core of soft delete!**

**Soft vs Hard Delete:**

```
objectscript

// ❌ Hard Delete (DON'T do this):
Do ##class(SEM.tblEmployee).%DeleteId(5)
// Permanently removes from database
// Cannot recover
// Breaks references

// ✅ Soft Delete (DO this):
Set employee = ##class(SEM.tblEmployee).%OpenId(5)
Set employee.deleteFlg = 1
Do employee.%Save()
// Still in database
// Just marked as deleted
// Can recover by setting deleteFlg = 0
```

**What happens after soft delete:**

objectscript

```
// Before delete:
ID: 5
EmployeeId: "12345"
Name: "山田太郎"
deleteFlg: 0  ← Active
upDateTime: "2024-12-20 10:00:00"

// After delete:
ID: 5
EmployeeId: "12345"
Name: "山田太郎"
deleteFlg: 1  ← Deleted!
upDateTime: "2024-12-25 14:30:00"  ← Updated

// All other data preserved!
```

### Why update upDateTime?

objectscript

```
// Track when record was deleted
// Useful for:
// - Audit logs
// - Data retention policies
// - Recovery decisions
// - Analytics

// Example: Delete employees marked as deleted > 7 years ago
SELECT * FROM SEM.tblEmployee
WHERE deleteFlg = 1
  AND upDateTime < (CURRENT_DATE - INTERVAL 7 YEAR)
```

**Line 34-41: Save and Respond**

```
objectscript

Set status = employee.%Save()

If $$$ISOK(status) {
  Set result.message = "Employee deleted successfully."
} Else {
  Set result.message = "Failed to delete employee."
  Set result.error = $System.Status.GetErrorText(status)
}
```

- Save changes (sets deleteFlg = 1)
- Return success/failure message

**Effect on queries:**

objectscript

```
// After soft delete, employee ID 5 invisible in:

// GetAllEmployees:
SELECT * FROM SEM.tblEmployee WHERE deleteFlg = 0
// ID 5 excluded

// GetEmployeeById:
Set emp = ##class(SEM.tblEmployee).%OpenId(5)
If (emp.deleteFlg = 1) { Return "Not found" }
// ID 5 treated as not found

// CreateEmployee duplicate check:
SELECT * FROM SEM.tblEmployee
WHERE EmployeeId = ? AND deleteFlg = 0
// Can reuse "12345" employee ID

// But data still in database!
// Can be recovered if needed
```

---

## Helper Method: Write JSON Response

objectscript

```
/// Helper method to write JSON response
ClassMethod WriteJSONResponse(data As %DynamicObject) As %Status
{
  // Set HTTP response headers
  Set %response.ContentType = "application/json"
  Set %response.CharSet = "UTF-8"
  Set %response.NoCharSetConvert = 1

  // Convert dynamic object to JSON string
  Set jsonString = data.%ToJSON()

  // Write to response stream
  Write jsonString

  Return $$$OK
}
```

**Line-by-Line Explanation:**

**Line 2: Method Signature**

```
objectscript

ClassMethod WriteJSONResponse(data As %DynamicObject) As %Status
```

- **%DynamicObject**: Type for dynamic objects
- Like JavaScript objects
- Can contain nested objects and arrays

**Line 4-6: Set HTTP Headers**

```objectscript
Set %response.ContentType = "application/json"
Set %response.CharSet = "UTF-8"
Set %response.NoCharSetConvert = 1
```

**%response:** Built-in variable for HTTP response

**ContentType:**

```objectscript
// Tells browser what type of data is being sent
Set %response.ContentType = "application/json"

// Other examples:
"text/html"          → HTML page
"text/plain"         → Plain text
"application/xml"    → XML data
"image/jpeg"         → JPEG image
```

**CharSet:**

```objectscript
```

```
// Character encoding
Set %response.CharSet = "UTF-8"

// UTF-8 supports:
// - English (ASCII)
// - Japanese (日本語)
// - Chinese (中文)
// - All Unicode characters
```

**NoCharSetConvert:**

```
objectscript

Set %response.NoCharSetConvert = 1

// 1 = Don't convert character encoding
// Data already in correct format (UTF-8)
// Prevents double conversion
```

**Line 8-9: Convert to JSON**

```
objectscript

Set jsonString = data.%ToJSON()
```

**%ToJSON() method:**

```
objectscript
```

```
// Dynamic object:
Set obj = {}
Set obj.name = "山田太郎"
Set obj.age = 30

// Convert to JSON string:
Set json = obj.%ToJSON()
// json = '{"name":"山田太郎","age":30}'

// Can now send this string in HTTP response
```

**Line 11-12: Write Response**

```
objectscript

Write jsonString
```

**Write statement:**

- Writes data to current output stream
- In web context: HTTP response body
- Similar to `console.log()` but for HTTP

**Complete flow:**

```
objectscript
```

```
// 1. Create response object:
Set result = {}
Set result.message = "Success"
Set result.data = empData

// 2. Call helper:
Do ..WriteJSONResponse(result)

// 3. Helper converts to JSON:
'{"message":"Success","data":{...}}'

// 4. Sets headers:
Content-Type: application/json
Charset: UTF-8

// 5. Writes to response:
Write json

// 6. Browser receives:
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{"message":"Success","data":{...}}
```

---

## Complete Request-Response Flow Example

Let me show you a complete end-to-end flow:

**Example: Update Employee**

1. Frontend (React):
   ↓
   axios.put('/sem/employee/5', {
    name: "佐藤花子",
    sex: 2,
    department: "技術部",
    ...
   })

2. HTTP Request:
   ↓
   PUT http://localhost:52773/sem/employee/5
   Content-Type: application/json

   {"name":"佐藤花子","sex":2,"department":"技術部",...}

3. IRIS receives request:
   ↓
   - URL matches: /employee/:id with PUT
   - Extracts: id = "5"
   - Calls: UpdateEmployee("5")

4. UpdateEmployee method:
   ↓
   - Validates id = "5"
   - Opens employee: %OpenId(5)
   - Parses JSON: %FromJSON(%request.Content)
   - Validates: name not empty, sex not empty
   - Updates properties:
    Set employee.Name = $ZCONVERT("佐藤花子", "I", "UTF8")

Set employee.Department = $ZCONVERT("技術部", "I", "UTF8")
Set employee.upDateTime = $ZDATETIME($HOROLOG, 3)
- Saves: employee.%Save()
- Creates response:
Set result.message = "Employee updated successfully."

5. WriteJSONResponse:
↓
- Sets headers: Content-Type, CharSet
- Converts to JSON: result.%ToJSON()
- Writes: '{"message":"Employee updated successfully."}'

6. HTTP Response:
↓
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{"message":"Employee updated successfully."}

7. Frontend (React):
↓
- axios resolves promise
- Shows success message
- Navigates to employee list
- Reloads data

## Summary: All REST API Endpoints

| Method | URL | Parameters | Purpose |
|---|---|---|---|
| POST | /signup | Body: name, email, password | Create user account |
| POST | /signin | Body: email, password | Authenticate user |
| GET | /employees | None | Get all active employees |
| GET | /employee/:id | URL: id | Get single employee |
| POST | /employee | Body: all employee fields | Create new employee |
| PUT | /employee/:id | URL: id, Body: employee fields | Update employee |
| DELETE | /employee/:id | URL: id | Soft delete employee |

---

## এই সাথে আপনার Backend এর সম্পূর্ণ explanation complete! 🎉

You now understand:

- ✅ Database classes (Persistent objects)
- ✅ REST API routing (XData UrlMap)
- ✅ All CRUD operations
- ✅ URL parameters
- ✅ JSON parsing and creation
- ✅ UTF-8 character handling

- ✅ Soft delete pattern
- ✅ Error handling
- ✅ Response formatting

**Ready for your presentation! 頑張ってください！🚀**