



## Complete Code Explanation - Part 7

### Backend (IRIS ObjectScript) - REST API Handler

---

#### 19. SEM.SEMRESTAPI.cls

**Purpose:** REST API handler যা frontend requests receive করে, process করে, এবং response পাঠায়।

This is the **MOST COMPLEX backend file!** It handles:

- HTTP request routing
- JSON parsing
- Database operations
- Error handling
- Response formatting

Let me break it down section by section:

```
objectscript
```

```

/// REST API for Employee Management System
Class SEM.SEMRESTAPI Extends %CSP.REST
{

/// URL Routing Map
XData UrlMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]
{
<Routes>
  <!-- Authentication APIs -->
  <Route Url="/signup" Method="POST" Call="AccountRegistration"/>
  <Route Url="/signin" Method="POST" Call="AccountLogin"/>

  <!-- Employee CRUD APIs -->
  <Route Url="/employees" Method="GET" Call="GetAllEmployees"/>
  <Route Url="/employee/:id" Method="GET" Call="GetEmployeeById"/>
  <Route Url="/employee" Method="POST" Call="CreateEmployee"/>
  <Route Url="/employee/:id" Method="PUT" Call="UpdateEmployee"/>
  <Route Url="/employee/:id" Method="DELETE" Call="DeleteEmployee"/>
</Routes>
}

```

## Understanding the Class Structure:

### Line 1-3: Class Definition

objectscript

```

/// REST API for Employee Management System
Class SEM.SEMRESTAPI Extends %CSP.REST
{

```

- **Extends %CSP.REST:** Built-in IRIS class for REST APIs
  - **%CSP:** Common Server Page (web framework)
  - **REST:** RESTful web service support
- Automatically handles:
  - HTTP request parsing
  - URL routing
  - Response formatting
  - Error handling

#### Line 5-19: URL Routing Map

```
objectscript  
  
XData UrlMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]  
{  
<Routes>  
  ...  
</Routes>  
}
```

- **XData:** Special block for embedded XML data
- **UrlMap:** Defines URL routes
- **XMLNamespace:** IRIS standard namespace

#### Route Structure:

```
xml
```

```
<Route Url="/path" Method="HTTP_METHOD" Call="ClassMethodName"/>
```

- **Url:** URL pattern to match
- **Method:** HTTP method (GET, POST, PUT, DELETE)
- **Call:** Which ClassMethod to execute

### Line 9-10: Authentication Routes

xml

```
<Route Url="/signup" Method="POST" Call="AccountRegistration"/>  
<Route Url="/signin" Method="POST" Call="AccountLogin"/>
```

### How routing works:

HTTP Request:

POST http://localhost:52773/sem/signup

↓

IRIS matches route: /signup + POST

↓

Calls: AccountRegistration() method

↓

Method processes request

↓

Returns JSON response

### Line 12-17: Employee CRUD Routes

xml

```
<Route Url="/employees" Method="GET" Call="GetAllEmployees"/>
```

- Get all employees list

xml

```
<Route Url="/employee/:id" Method="GET" Call="GetEmployeeById"/>
```

- **:id**: URL parameter (variable)
- Example: `/employee/42` → id = 42
- Get specific employee

xml

```
<Route Url="/employee" Method="POST" Call="CreateEmployee"/>
```

- Create new employee

xml

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

- Update existing employee
- Example: `/employee/42` → Update employee 42

xml

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

- Delete employee (soft delete)
  - Example: `/employee/42` → Delete employee 42
- 

## Authentication Methods

### Method 1: Account Registration (Signup)

objectscript

```
/// Account registration endpoint
ClassMethod AccountRegistration() As %Status
{
    // Initialize response
    Set result = {}
    Set result.message = ""

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

        // Extract input data
        Set reqName = requestObject.inputName
        Set reqEmail = requestObject.inputEmail
        Set reqPassword = requestObject.inputPassword

        // Validation: Check required fields
        If (reqEmail = "") {
            Set result.message = "Email is required."
            Do ..WriteJSONResponse(result)
            Return $$$OK
        }

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

        // Check if email already exists
        Set sqlQuery = "SELECT ID FROM SEM.tblAccount WHERE Email = ?"
        Set statement = ##class(%SQL.Statement).%New()
```

```
Set status = statement.%Prepare(sqlQuery)
Set resultSet = statement.%Execute(reqEmail)

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

// Create new account
Set newAccount = ##Class(SEM.tblAccount).%New()
Set newAccount.Email = reqEmail
Set newAccount.Name = $ZCONVERT(reqName, "I", "UTF8")
Set newAccount.Password = reqPassword

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

If $$$ISOK(status) {
    Set result.message = "Registration successful."
    Set result.id = newAccount.%Id()
} Else {
    Set result.message = "Registration failed."
}

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

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

## Line-by-Line Explanation:

### Line 1-2: Method Declaration

```
objectscript  
  
/// Account registration endpoint  
ClassMethod AccountRegistration() As %Status
```

- **ClassMethod:** Static method (doesn't need object instance)
- **AccountRegistration:** Method name (matches Route Call)
- **As %Status:** Return type (status code)

### Line 4-6: Initialize Response

```
objectscript  
  
Set result = {}  
Set result.message = ""
```

- **{}**: Creates empty dynamic object (like JavaScript object)
- **result.message:** Property assignment
- Will be converted to JSON later

### ObjectScript Dynamic Objects:

```
objectscript
```

```
// JavaScript:
let result = {
  message: "",
  data: null
};

// ObjectScript:
Set result = {}
Set result.message = ""
Set result.data = ""
```

#### Line 8-14: Parse Request Body

```
objectsript

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

  Set reqName = requestObject.inputName
  Set reqEmail = requestObject.inputEmail
  Set reqPassword = requestObject.inputPassword
```

#### Breaking it down:

**%request.Content:** Built-in variable containing HTTP request body

```
json
```

```
{  
  "inputName": "田中太郎",  
  "inputEmail": "tanaka@example.com",  
  "inputPassword": "password123"  
}
```

**%FromJSON():** Parses JSON string to dynamic object

```
objectscript  
  
// JSON string → Dynamic object  
Set jsonString = '{"name":"John","age":30}'  
Set obj = ##Class(%DynamicAbstractObject).%FromJSON(jsonString)  
  
// Access properties:  
Write obj.name // "John"  
Write obj.age  // 30
```

**Extract values:**

```
objectscript  
  
Set reqName = requestObject.inputName  
// reqName = "田中太郎"  
  
Set reqEmail = requestObject.inputEmail  
// reqEmail = "tanaka@example.com"  
  
Set reqPassword = requestObject.inputPassword  
// reqPassword = "password123"
```

## Line 16-22: Validate Email

```
objectscript

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

### Syntax explained:

- **If (condition):** Conditional statement
- **reqEmail = "":** Check if empty
- **Do ..Method():** Call instance method
- **..:** Reference to current class instance (like `this` in JS)
- **Return \$\$\$OK:** Return success status

## Line 24-29: Validate Password

```
objectscript

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

- Same validation pattern for password

### Line 31-40: Check Duplicate Email

```
objectscript

// SQL query with parameter placeholder
Set sqlQuery = "SELECT ID FROM SEM.tblAccount WHERE Email = ?"

// Create statement object
Set statement = ##class(%SQL.Statement).%New()

// Prepare query (compile SQL)
Set status = statement.%Prepare(sqlQuery)


// Execute with parameter
Set resultSet = statement.%Execute(reqEmail)


// Check if any row returned
If resultSet.%Next() {
    Set result.message = "Email already exists."
    Do ..WriteJSONResponse(result)
    Return $$$OK
}
```

### SQL in ObjectScript:

#### Parameterized Query (Safe from SQL Injection):

```
objectscript
```

```
//  Good (parameterized):
Set sql = "SELECT * FROM table WHERE email = ?"
Set resultSet = statement.%Execute(userInput)
// userInput is escaped automatically

//  Bad (SQL injection risk):
Set sql = "SELECT * FROM table WHERE email = '" & _userInput & "'"
// If userInput = "; DROP TABLE users; --"
// SQL becomes: SELECT * FROM table WHERE email = "; DROP TABLE users; --'
```

### **resultSet.%Next():**

```
objectscript

// Returns true if row exists, false if no rows
If resultSet.%Next() {
    // Email found in database
    // Get the ID:
    Set id = resultSet.%Get("ID")
}
```

### **Line 42-46: Create New Account**

```
objectscript

Set newAccount = ##Class(SEM.tblAccount).%New()
Set newAccount.Email = reqEmail
Set newAccount.Name = $ZCONVERT(reqName, "I", "UTF8")
Set newAccount.Password = reqPassword
```

**%New():** Creates new object instance

objectscript

// JavaScript:

let account = new Account();

// ObjectScript:

Set account = ##Class(SEM.tblAccount).%New()

## **\$ZCONVERT(): Character encoding conversion**

objectscript

Set newAccount.Name = \$ZCONVERT(reqName, "I", "UTF8")

//                    ↓        ↓ ↓ ↓ ↓

//                    Function | | | Target encoding

//                    Value | Direction

//                    "I" = Input (external → internal)

## **Why \$ZCONVERT?**

objectscript

```
// Japanese characters from HTTP request:
reqName = "田中太郎" (UTF-8 from JSON)

// IRIS internal format might be different
// $ZCONVERT ensures proper encoding

// Without conversion:
Set account.Name = "田中太郎"
// Might display as: ç°ä,åªé (mojibake!)

// With conversion:
Set account.Name = $ZCONVERT("田中太郎", "I", "UTF8")
// Displays correctly: 田中太郎 ✓
```

### Directions:

```
objectscript

// "I" = Input (external → IRIS internal)
$ZCONVERT(data, "I", "UTF8")

// "O" = Output (IRIS internal → external)
$ZCONVERT(data, "O", "UTF8")
```

### Line 48-55: Save and Respond

```
objectscript
```

```
Set status = newAccount.%Save()

If $$$ISOK(status) {
    Set result.message = "Registration successful."
    Set result.id = newAccount.%Id()
} Else {
    Set result.message = "Registration failed."
}
```

**%Save():** Persists object to database

```
objectscript

// Returns status object
Set status = newAccount.%Save()

// Check if successful:
If $$$ISOK(status) {
    // Success!
} Else {
    // Failure - get error message:
    Set errorMsg = $System.Status.GetErrorText(status)
}
```

**\$\$\$ISOK():** Macro to check status

```
objectscript
```

```
// Macro: $$$ISOK(status)
// Expands to: $$$ISOK(status) = 1
```

```
// Similar macros:
```

```
$$$OK      → Success status
```

```
$$$ERROR   → Create error status
```

```
$$$ISERR   → Check if error
```

**newAccount.%Id():** Gets database ID after save

```
objectscript
```

```
// Before save:
```

```
newAccount.%Id() → ""
```

```
// After save:
```

```
newAccount.%Id() → "5" (or whatever ID was assigned)
```

### Line 57-59: Exception Handling

```
objectscript
```

```
} Catch ex {
```

```
    Set result.message = "Error: " _ex.DisplayString()
```

```
}
```

- **Catch ex:** Catches any exception
- **ex.DisplayString():** Gets error message
- **"\_":** String concatenation operator

## String Concatenation in ObjectScript:

```
objectscript

// JavaScript:
let msg = "Error: " + error.message;

// ObjectScript:
Set msg = "Error: " _ error.DisplayString()

// Multiple concatenations:
Set fullName = firstName _ " " _ lastName
// "John" _ " " _ "Doe" → "John Doe"
```

## Line 61-63: Send Response

```
objectscript

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

- **WriteJSONResponse()**: Built-in method (we'll see implementation later)
  - Converts result object to JSON
  - Sets HTTP headers
  - Writes to response stream
-

## Method 2: Account Login (Signin)

objectscript

```
/// Account login endpoint
ClassMethod AccountLogin() As %Status
{
    Set result = {}
    Set result.message = ""

    Try {
        // Parse request
        Set requestObject = ##Class(%DynamicAbstractObject).%FromJSON(%request.Content)
        Set reqEmail = requestObject.inputEmail
        Set reqPassword = requestObject.inputPassword

        // Validation
        If (reqEmail = "") {
            Set result.message = "Email is required."
            Do ..WriteJSONResponse(result)
            Return $$$OK
        }

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

        // Find account by email
        Set sqlQuery = "SELECT ID, Password FROM SEM.tblAccount WHERE Email = ?"
        Set statement = ##class(%SQL.Statement).%New()
        Do statement.%Prepare(sqlQuery)
        Set resultSet = statement.%Execute(reqEmail)

        // Check if account exists
```

```

If resultSet.%Next() {
    Set accountId = resultSet.%Get("ID")
    Set storedPassword = resultSet.%Get("Password")

    // Verify password
    If (storedPassword = reqPassword) {
        Set result.message = "Authentication successful."
        Set result.id = accountId
    } Else {
        Set result.message = "Invalid password."
    }
} Else {
    Set result.message = "Email is not registered."
}

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

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

```

### Key Differences from Registration:

#### Line 26: Query includes Password

objectscript

```
Set sqlQuery = "SELECT ID, Password FROM SEM.tblAccount WHERE Email = ?"
```

- Need to retrieve password for comparison

- Registration only checks if email exists (SELECT ID)

### Line 32-35: Extract account data

```
objectscript

If resultSet.%Next() {
    Set accountId = resultSet.%Get("ID")
    Set storedPassword = resultSet.%Get("Password")
}
```

### resultSet.%Get(columnName):

```
objectscript

// Get column value from result set
Set id = resultSet.%Get("ID")
Set password = resultSet.%Get("Password")

// Column names are case-insensitive:
resultSet.%Get("ID") = resultSet.%Get("id")
```

### Line 37-42: Password Verification

```
objectscript

If (storedPassword = reqPassword) {
    Set result.message = "Authentication successful."
    Set result.id = accountId
} Else {
    Set result.message = "Invalid password."
}
```

## Simple comparison:

```
objectscript

// Current (plaintext):
If (storedPassword = reqPassword) {
  // Passwords match
}

// Production (hashed):
If ($System.Encryption.BCrypt.Verify(reqPassword, storedPassword)) {
  // Password verified against hash
}
```

## Security Note:

```
objectscript

// ❌ Current implementation (INSECURE):
// - Stores passwords in plaintext
// - Simple string comparison
// - No rate limiting
// - No account lockout

// ✅ Production should have:
// - BCrypt password hashing
// - Rate limiting (prevent brute force)
// - Account lockout after failed attempts
// - Session tokens (JWT)
// - HTTPS only
```

---

## Employee CRUD Methods

### Method 3: Get All Employees

```
objectscript
```

```

/// Get all employees (active only)
ClassMethod GetAllEmployees() As %Status
{
    Set result = {}
    Set result.employees = []

    Try {
        // SQL query for active employees
        Set sqlQuery = "SELECT ID, EmployeeId, Name, KanaName, Sex, "_
            "PostCode, Address, PhoneNumber, Department, "_
            "RetireFlg, upDateTime "_
            "FROM SEM.tblEmployee "_
            "WHERE deleteFlg = 0 "_
            "ORDER BY upDateTime DESC"

        Set statement = ##class(%SQL.Statement).%New()
        Set status = statement.%Prepare(sqlQuery)
        Set resultSet = statement.%Execute()

        // Loop through results
        While resultSet.%Next() {
            Set employee = {}
            Set employee.id = resultSet.%Get("ID")
            Set employee.EmployeeId = resultSet.%Get("EmployeeId")
            Set employee.Name = $ZCONVERT(resultSet.%Get("Name"), "O", "UTF8")
            Set employee.KanaName = $ZCONVERT(resultSet.%Get("KanaName"), "O", "UTF8")
            Set employee.Sex = resultSet.%Get("Sex")
            Set employee.PostCode = resultSet.%Get("PostCode")
            Set employee.Address = $ZCONVERT(resultSet.%Get("Address"), "O", "UTF8")
            Set employee.PhoneNumber = resultSet.%Get("PhoneNumber")
            Set employee.Department = $ZCONVERT(resultSet.%Get("Department"), "O", "UTF8")
            Set employee.RetireFlg = resultSet.%Get("RetireFlg")
        }
    }
}

```

```
Set employee.upDateTime = resultSet.%Get("upDateTime")

// Add to array
Do result.employees.%Push(employee)
}

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

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

### Line-by-Line Explanation:

#### Line 4-5: Initialize Response

```
objectscript

Set result = {}
Set result.employees = []
```

- `[]`: Creates empty dynamic array
- Will hold array of employee objects

### Dynamic Array in ObjectScript:

```
objectscript
```

```
// JavaScript:
let arr = [];
arr.push({name: "John"});
arr.push({name: "Jane"});

// ObjectScript:
Set arr = []
Do arr.%Push({}%Set("name", "John"))
Do arr.%Push({}%Set("name", "Jane"))
```

### Line 8-13: Multi-line SQL Query

```
objectscript

Set sqlQuery = "SELECT ID, EmployeeId, Name, KanaName, Sex, "_
               "PostCode, Address, PhoneNumber, Department, "_
               "RetireFlg, upDateTime "_
               "FROM SEM.tblEmployee "_
               "WHERE deleteFlg = 0 "_
               "ORDER BY upDateTime DESC"
```

### Multi-line string concatenation:

```
objectscript

// " _ " at end of line continues on next line
Set str = "First line "_
         "Second line "_
         "Third line"

// Result: "First line Second line Third line"
```

**WHERE deleteFlg = 0:**

- Only active employees (not soft-deleted)
- Deleted employees (deleteFlg = 1) excluded

**ORDER BY upDateTime DESC:**

- **DESC:** Descending order (newest first)
- Most recently updated employees appear first

**Line 20-34: Loop Through Results**

```
objectscript  
  
While resultSet.%Next() {  
    Set employee = {}  
    Set employee.id = resultSet.%Get("ID")  
    // ... set all properties  
  
    Do result.employees.%Push(employee)  
}
```

**While loop:**

```
objectscript
```

```
// %Next() returns:  
// - True if more rows available  
// - False if no more rows  
  
While resultSet.%Next() {  
    // Process current row  
}  
  
// Similar to JavaScript:  
while (resultSet.next()) {  
    // Process row  
}
```

#### Line 24: \$ZCONVERT for Output

```
objectscript  
  
Set employee.Name = $ZCONVERT(resultSet.%Get("Name"), "O", "UTF8")
```

- **"O"**: Output direction (IRIS internal → UTF-8)
- Ensures Japanese characters sent correctly to frontend

#### Why "O" here but "I" on input?

```
objectscript
```

```
// Input (saving to database):  
Set account.Name = $ZCONVERT(jsonData, "I", "UTF8")  
// External UTF-8 → IRIS internal format  
  
// Output (reading from database):  
Set jsonName = $ZCONVERT(account.Name, "O", "UTF8")  
// IRIS internal format → External UTF-8
```

### Line 35: Add to Array

```
objectscript  
  
Do result.employees.%Push(employee)
```

- **%Push()**: Adds element to end of array
- Like JavaScript's `array.push()`

### Final result structure:

```
json
```

```
{
  "employees": [
    {
      "id": 1,
      "EmployeeId": "12345",
      "Name": "山田太郎",
      "Sex": 1,
      ...
    },
    {
      "id": 2,
      "EmployeeId": "67890",
      "Name": "佐藤花子",
      "Sex": 2,
      ...
    }
  ]
}
```

---

I'll continue with the remaining CRUD methods in the next section. These get more complex with URL parameters and UPDATE/DELETE operations!

Should I continue with:

- GetEmployeeById (URL parameters)
- CreateEmployee (validation + save)
- UpdateEmployee (load + modify + save)
- DeleteEmployee (soft delete)

- WriteJSONResponse helper method

?