



VoiceConsole 3.0

Web Services

Operator Methods

- VoiceConsole Sample Data
- VoiceConsoleServices class overview
 - Add necessary code to build calls
 - Identify locations in our code for hooks
- Hook into VoiceConsole methods
 - createOperator
 - deleteOperator
 - fetchOperatorByID
 - fetchAllOperators
 - modifyOperatorID

- Give ourselves something to work with
- Easiest methods to implement
 - Fetching
 - Deleting
 - Modifying the ID
- Create a few operators in the VoiceConsole interface
 - Use the “Unassigned” site

- Some information as private variables
 - URLs, URIs and login information
- Every call starts the same way
 - setUpCall() method
 - Exception handling
- Need to be specific with parameters
 - SOAP-ENC translation cannot happen behind the scenes; need to use XSD types
 - Use the guide and sample code to determine best data types to use

- Private static variables
 - soapURL
 - userID and password
 - operatorURI, terminalURI, and taskURI
- The setUpCall() method
- Looking for hooks
 - Match services to model methods
 - Usually in EmployeeDao implementation
 - This is where data is stored or retrieved in our app
 - Keeps the mojo separate from the entity and action classes

- Create the code in VoiceConsoleServices
 - src\main\java\com.mycompany.wsdemo...
 - Sends the SitePK for the site to query
 - Returns a List of HashMaps
- Add EmployeeAction variable and method
 - List<HashMap> operators + getters / setters
 - public String listVC()
- Create a new Employee view
 - src\main\webapp\jsp\employee\listVC.jsp
- Add the action in Struts.xml
 - src\main\resources\struts.xml
- Add a link to the action in employee\list.jsp

- Create the code in VoiceConsoleServices
 - Sends the OperatorID and SitePK
 - Returns a HashMap
- Add to EmployeeDaoImpl.save() method
 - `src\main\java\com.mycompany.wsdemo.dao`
 - Match on `employee.username`
 - Check to see if the employee already exists before creating it in VoiceConsole
- Use a breakpoint to see it work
 - `src\main\webapp\jsp\employee\listVC.jsp`

- Create the code in VoiceConsoleServices
 - Check guide for expected inputs
 - Be sure to use the right XMLType for each parameter
 - Returns the created operator's PK (an Integer)
- Add to EmployeeDaoImpl.save() method
 - After checking if the employee exists
 - Combine firstName and lastName for fullName
 - Convert to lowercase for spokenName
 - Use the employeeld as the Pin
- Use a breakpoint to see it work
- Check the results in VoiceConsole

- Create the code in VoiceConsoleServices
 - Expects operatorID and newOperatorID
 - Returns operatorPK of the affected operator
- Add to EmployeeDaoImpl.save() method
 - Place before the fetch, since fetch is based on operatorID
 - Employee is the new one, so use getByld() to access previous version (before actually saving it)
 - We do not care much for the results since we will go ahead and pull the operator in the next step
- Use a breakpoint to see it work
- Check the results in VoiceConsole

- Create the code in VoiceConsoleServices
 - Expects operatorPK for an input
 - We will find it by using fetchOperatorByID
 - Returns nothing (void)
- Add to EmployeeDaoImpl.delete() method
 - Get the operatorPK value for that employee
 - Delete from VoiceConsole first
 - If something goes drastically wrong, the record will remain so we can try again
- Use a breakpoint to see it work
- Check the results in VoiceConsole

- Sending messages
 - Creating teams
 - Voice templates
 - Devices
 - Tasks
 - Task Packages
-
- Model the data, create views and provide data for templates using action controllers

- Not all VC methods have WS interfaces
 - Modify other properties of an operator
 - Delete a team
 - Delete a device
 - Connect a device with an operator and a task package
- A Secret Clue!
 - it is *possible* to use most if not all methods you can imagine VoiceConsole offers, but anything outside this list is undocumented and unsupported (for now)

Questions and Answers

