# Washington
# Commercial Groundfish Fisheries
# Data Collection and Processing

by

Tien-Shui Tsou
Phillip M. Weyland
Mariko Langness

Washington Department of Fish and Wildlife
Fish Program
Fish Management Division – Marine Fish Science

Fish Program Report Number FPT 15-07

# Abstract

Washington Department of Fish and Wildlife (WDFW) supports groundfish stock assessments and management of fisheries through several interrelated groups that collect and process biological and catch data. The commercial components of these data are collected and processed from four data sources: fish receiving tickets, fisheries logbooks, species composition sampling, and biological sampling. This report describes current WDFW commercial groundfish fisheries data processing procedures.

# Table of Contents

# List of Tables

# List of Figures

# List of Appendices

# Introduction

Fishery managers rely on timely and accurate data to effectively manage the resources and industry. In Washington state, groundfish fishery management is accomplished through state, federal and tribal management teams governed by state commission members and interstate council representatives. Washington state policy and direction for fish species are established by the Washington Fish and Wildlife Commission (FWC or Commission). The Commission, comprised of nine citizen members, has legislated authority to establish the basic rules and regulations governing the time, place, manner, and methods used to harvest fish in Washington state waters (0-3 miles). Washington coastal fisheries management is coordinated with the National Marine Fisheries Service (NMFS) through the Pacific Fishery Management Council (PFMC or Council). The Council, comprised of 14 voting representatives, governs the commercial and recreational fisheries in federal waters (3-200 miles) off Washington, Oregon, and California. In addition, Washington coastal tribes co-manage fisheries in federal waters with NMFS, and in state waters with the Washington Department of Fish and Wildlife (WDFW).

The Pacific Fishery Management Council manages fisheries for about 119 species of groundfish (also known as bottomfish), salmon, coastal pelagic species (sardines, anchovies, and mackerel), and highly migratory species (tunas, sharks, and swordfish). The Council provides annual fishing quotas and regulations designed to achieve optimum yield from a fishery, and such management measures are implemented by NMFS. Management teams comprised of state and NMFS representatives prepare fishery management plans for PFMC, utilizing their respective fishery monitoring programs, and completed stock assessments and impact analyses. Timely and accurate fisheries data are essential to the completion of these assessments and the effective management of fisheries.

Management teams and research scientists obtain fisheries-dependent monitoring data from two data networks managed by the Pacific States Marine Fisheries Commission (PSMFC): the Pacific Fisheries Information Network (PacFIN) for commercial fisheries and the Recreational Fisheries Information Network (RecFIN) for recreational fisheries. These regional fisheries data networks are the products of a joint federal and state project focused on fisheries data collection and information management; providing fisheries data from Washington, Oregon, California, Alaska, and British Columbia. PSMFC is an interstate compact agency that helps resource agencies and the fishing industry sustainably manage valuable Pacific Ocean resources in five states: Washington, Oregon, California, Alaska, and Idaho.

Washington Department of Fish and Wildlife groundfish stock assessments and management of commercial and recreational fisheries are supported through several interrelated groups that

collect and process biological and catch data. Details on the systematic flow of commercial data through these groups are shown in Figure 1. This report focuses on the data processing groups, summarizing current WDFW commercial groundfish fisheries data processing procedures. Data processing procedures performed by PacFIN are not within the scope of this report. Data collection methods are summarized in Tsou et al. (2015) and detailed protocols are provided in the Coastal Commercial Groundfish Port Sampling Program – Procedures Manual (Port Sampling Manual). Historical sampling protocols and data processing procedures are summarized in Tagart (1997) and Konkel (1998).

Four types of data are collected from commercial groundfish fisheries operating in Washington State: fish receiving tickets, fisheries logbooks, species compostions, and biological data. Each data source provides essential information needed to effectively monitor and manage groundfish fisheries. Landed catch data are retrieved from fish receiving tickets, and fishing effort and location data are retrieved from fisheries logbooks. Species composition sampling provides an estimate of species landed in single and mixed-species "market" categories (management unit in which a single species or multiple species comprise a category based on management needs and ease or difficulty in speciating). Biological data provide length, weight, sex, maturity, and age data for all groundfish samples.

# Fish Receiving Tickets

Fish receiving tickets are an official document provided at no cost to fish dealers for commercial catch accounting and tax revenue purposes. Dealers are required by state regulation (WAC 220-69-280) to accurately and completely fill out these documents for all commercial fisheries. The primary information obtained from fish tickets are total landed weight and value of each market category. The landed catch data are reported via two ticket pathways: 1) paper tickets (Figure 2); and 2) electronic tickets (e-tix). Currently, the groundfish Individual Fishing Quoata (IFQ) fishery is the only fishery required to use the e-ticket submission format (Figure 3). Landed catch data from all other fisheries are reported through paper tickets, but several are in the process of switching to e-tix.

## Data Collection

The marine fish receiving ticket captures market category landed, scale weight of market category, dressed status, gear used, WDFW management area (Figure 4) where most of the fish were caught, landing date, seller and dealer information, along with economic data such as the price per pound paid for each market category. Most marine fisheries allow partial offloads, and vessels can land their catch over multiple days and at multiple ports, sometimes in different states from one trip. Therefore, multiple tickets can be generated by a dealer for each landing. Some marine fisheries require that once an offload commences all catch be offloaded before the vessel leaves the dock.

Dealers submit marine fish paper tickets to port samplers at regional offices (via email or obtained at port) and to the WDFW Commercial Harvest Data Team (Data Team) at headquarters (via mail) no later than the 6$^{th}$ working day after landing date (WAC 220-69-260). Dealers participating in tribal fisheries send a fish ticket copy to WDFW and the Northwest Indian Fisheries Comission (NWIFC) to be entered into NWIFC's Tribal Online Catch Account System (TOCAS). These tribal landings are also required to be reported to WDFW by the 6$^{th}$ working day after landing date. The Data Team keypunches data and perfoms double data entry (QA/QC) into the Washington License and Fish receiving Ticket (LiFT) database (Morningstar, pers. comm.). E-tix (IFQ tickets) are tracked in real time and reported via web user interface to the PSMFC e-tix system within 24-hours of a landing. E-tix are reviewed and then manually released into LiFT. Both paper tickets and e-tix are compiled by port samplers to provide unofficial fish ticket data ("soft catch data") to produce weekly reports, also known as the "market report". Market reports are sent to PacFIN for its Quota Species Monitoring (QSM), a "real time" summary of catch data for limited entry, open access, and tribal groundfish fleets.

## Data Processing

After the raw landed catch data are entered into the LiFT database, other information, such as port of landing and round pound (whole fish) weights, is generated by the system automatically. Actual port of landing is not required on Washington fish receiving tickets and "port" is instead generated based on the dealer business address stored in the database. This long-time approach no longer works under current fish dealer operation patterns, and WDFW is proposing rule changes to resolve this issue.

Fish are often not landed in whole (round) condition, and may be filleted, gutted, or headed (dressed). To estimate round weights, conversion factors (Table 1) are applied to dressed weights reported on fish tickets. Conversion factors are based on Crapo et al. (1993) or historical field studies (Stanley, pers. comm.). After these calculations and conversions, the fish ticket data in the LiFT database are transmitted to PacFIN via a monthly feed access application using an SFTP client. The PacFIN data feed application (Appendix A) further converts WDFW codes into PacFIN specification. Figure 5 demonstrates the fish ticket data flow.

# Coastal Trawl Logbook / Area Composition

Logbooks provide fishery operation information such as fishing location and effort. In Washington, coastal groundfish trawlers are required by state regulation (WAC 220-44-080) to maintain and submit a complete and legible logbook containing data for each tow of a fishing trip. WDFW uses the standardized Washington-Oregon-California (W-O-C) trawl logbooks jointly developed by W-O-C state agencies and PFMC (Figure 6). The main information generated from the logbook data are the spatial distribution of the catch by market category, also referred to as area composition. The biggest challenge in generating area composition is matching a logbook for each trip to associated landing(s)/fish ticket(s) and resolving the differences between the two data sources.

In addition to coastal groundfish trawl logbooks, WDFW collects and processes logbooks for several coastal fisheries: Pink Shrimp, Spot Shrimp, Dungeness Crab, Hagfish, and charter fisheries; and Puget Sound fisheries. These logbooks are not part of the PacFIN data flow.

## Data Collection

Vessels using trawl gears off Washington coastal waters are required to have complete logbooks containing data for each tow of the trip, including set date, time, latitude and longitude, and hailed market category weight. These logbooks are collected from vessel captains by port samplers and entered into the coastal trawl logbook system (CTLS) database at their respective duty stations via front-end Access forms (Appendix D). Logbooks not collected by port samplers are mailed to local regional offices. Port samplers match logbooks to fish tickets based on vessel and return date information and enter this into a CTLS data entry Access form. After this, the data are immediately available in the CTLS database to be used in the PacFIN data feed application by the database administrator (Appendix A).

## Data Processing

The structure of the WDFW database, specifically to logbooks, is very similar to the PacFIN database, which allows us to send the logbook data feed with minor conversions and processing. Most of the data feed content is primary data which are raw data collected, not calculated. The entire process including SQL scripts and Microsoft Access macros and sub routines are described in the PacFIN data feed system documentation (Appendix A). Figure 7 demonstrates the trawl logbook data flow.

Important processes occuring through the data feed sytem include relating logbook transactions, conversion of management areas, reconciliation of hailed logbook weights to fish ticket landing weights, and calculation of area compositions.

### ***Logbook Transactions***

An important aspect of the data feed is relating the transaction types, particularly types 1 - trip data, 2 - tow data, and 3 - catch data (Table 2). Keys are created which uniquely identify record within the transaction type. The keys created for trip data transactions are a composite built from federal document number (identifying vessel) and the depart date. Since for some fisheries WDFW allows multiple fish tickets per landing, and catch from one trip can be landed at multiple ports, there is no method at this point to truly identify a unique trip. The occurrence of a vessel having multiple trips with the same depart date is expected to be very small. However, caution should be taken if analyzing logbook information on the trip level, for what is identified as a single trip can contain tows from multiple trips (PacFIN tables: lbk_trip, lbk_tow, lbk_catch).

### ***Management Area Processing***

WDFW reports PMFC area to PacFIN, but tow locations are originally recorded in the CTLS by latitude and longitude coordinates which are converted to WDFW management area using a SQL stored procedure, and then to PMFC area using the same procedure (Table 3). Note that this WDFW area definition is different from the catch area definition used with fish tickets. Current fish ticket catch areas do not align with PFMC area.

### ***Fish Ticket Adjusted Logbook Pounds Processing***

In order to provide correct area composition of groundfish catch, hailed weights recorded in logbooks (landing weights estimated by skippers during a fishing trip) are adjusted to match fish ticket landing weights. Prior to 2005, WDFW used algorithms developed by department staff (Clark 1986 a & b; Konkel 1998) for logbook-fish ticket reconciliation. To promote data standardization across the states, WDFW started using PacFIN specifications in 2005. Though PacFIN specifications are provided to all reporting state agencies, the process used to produce data to PacFIN specifications varies slightly across Washington, Oregon, and California. Unlike other reporting agencies, WDFW does not send fish ticket adjusted tow weight along with the logbook data feed. PacFIN does this adjustment with scripting and logic developed in their database. The Oregon Department of Fish and Wildlife (ODFW) and California Department of Fish and Wildlife (CDFW) calculate the adjusted pounds using their own script and send it along with their data feed. Though the scope of this report pertains to WDFW data processes, this is a

PacFIN requirement that WDFW should provide, so the PacFIN adjustment process is described below.

The PacFIN SQL program (lbk_set_apounds.sql) is used to relate fish tickets to logbooks and then distribute the related fish ticket weight over related logbook tows. The goal is to provide the most accurate market category landed weight with the most accurate spatial distribution.

WDFW provides logbook data with related fish tickets (Table 2, transaction type 4) to PacFIN. For logbooks that do not have a related fish ticket explicitly provided by WDFW, PacFIN determines implicit fish ticket-logbook relationship by vessel and fish ticket date. If the fish ticket date is between the logbook depart and return date, these tickets are implicitly related to the logbook trip key. If the fish ticket was created after the logbook return date, PacFIN cannot determine fish ticket and logbook relationships.

PacFIN sums market category by trip and distributes related fish ticket landed weight according to proportion of tow hailed weights to sum of species within same trip. For fish ticket landed weights that do not appear in related logbook tows, the weight is evenly distributed across all tows without regard to total hailed weight of all species. If unable to calculate adjusted pounds, PacFIN sets adjusted pounds to the hailed weight and identifies such cases in the database.

***Area Composition***

Area compositions (AreaComps) are created to distribute fish ticket landed weight by market category across PMFC management areas. Since fish tickets only record the area from which the majority of the fish was caught for the entire trip, area compositions build proportions based on trawl logbook data which have specific location information on a tow-by-tow level with which the fish ticket weight can be distributed over.

AreaComps are exclusively built from trawl logbook data, specifically using the tow adjusted pounds that are produced from the fish ticket logbook reconciliation process and PMFC area. The logbook data used in creating these AreaComps are first extracted from PacFIN logbook tables which are populated by WDFW through the logbook data feed. This allows WDFW to use the previously formatted PacFIN port codes and fish ticket adjusted logbook weights calculated by PacFIN. Although AreaComp proportions can be calculated for each trawl logbook, they are calculated at the stratum level for each market category. These proportions are then sent to PacFIN.

WDFW has defined the strata it uses to calculate AreaComps for market categories as trip type, port, and month. Trip type is defined as Puget Sound or coastal. Port includes all individual ports previously sent in the logbook data feed to PacFIN (Table 4, PacFIN Port Code).

For a given market category, the proportion *Y* caught in PMFC area *a*, as trip type *t*, within month *m*, at port *p* is calculated as:

$$Y_{tmpa} = \frac{\sum_h W_{tmpah}}{\sum_a \sum_h W_{tmpah}}$$

where, $W_{tmpah}$ is market category weight in PMFC area *a* caught in haul *h*.

These AreaComps do not get applied to other gear types in PacFIN data processing, specifically when creating the vdrfd table utilized by PacFIN's online Explorer Query Tool. Fish tickets in this table that use other gear types do not have AreaComps applied and must use the WDFW catch area recorded in the fish ticket data feed. New WDFW catch areas (Figure 4) do not align well with PFMC areas, but align well with International North Pacific Fisheries Commission (INPFC) areas. If querying WDFW data from PacFIN by PFMC area, verify total weights by INPFC areas to insure no weight is being dropped.

# Rockfish Species Composition

## Data Collection

Rockfish species composition (SpComp) data are collected to estimate the catch of various species landed in single and mixed-species market categories. SpComp samples are routinely taken from mixed species market categories (Slope, Shelf, and Nearshore Rockfish), and in some cases from single-species market categories (Darkblotched Rockfish, Shortspine and Longspine Thornyhead) to verify fishermen/fish dealers sorting accuracy (Table 5). Sampling is stratified by port per quarter and based on landed weights for each market category. Further details on data collection methods and protocols are summarized in Tsou et al. (2015) and the Port Sampling Manual.

## Data Processing

Collected species composition data are emailed to a WDFW IT specialist to perform data entry and QA/QC. Data are entered into the SpComp database using an Access interface. Currently, all data are recorded on a paper form, but a data collection application (iPad interface) is in the testing phase. After data are entered, the proportions of species are calculated at the stratum level for each market category; though, proportions can be calculated for each sample and applied to the related landing. Calculated proportions are transmitted to PacFIN quarterly using the PacFIN data feed application (Appendix A). Figure 8 demontrates the SpComp data flow.

The strata WDFW has defined for its SpComp sampling is quarter, gear group, and port group (Table 5). We assume that fish tickets belonging to each unique stratum will have a similar species composition. A quarter is defined as calendar quarter; Jan-Mar is Quarter 1, Apr-Jun is Quarter 2, Jul-Sep is Quarter 3, and Oct-Dec is Quarter 4. The gear group is defined as Trawl or Hook-and-Line. The Trawl gear group includes bottom trawl (small footrope), selective flatfish, midwater trawl, and roller trawl gear. The Hook-and-Line gear group includes set line gear.

There are four major port groups: Westport, Neah Bay, Bellingham, and Other (Table 4). These port groups contain individual ports proximal to the major port. The Westport port group contains Ilwaco, Westport, and other smaller proximal ports. The Neah Bay port group contains Neah Bay and La Push ports. The Bellingham port group contains Bellingham, Blaine, Seattle, and other smaller proximal ports. The Other group contains all ports in Alaska, California, and Oregon.

For a given market category, the proportion of species *s* caught by gear *g* in quarter *q* within port group *p* is calculated as

$$X_{gpsq} = \frac{\sum_n W_{gpsqn}}{\sum_s \sum_n W_{gpqsn}}$$

where, $W_{gpsqn}$ is the weight of species *s* in sample *n*.

If there is no sample taken in a quarter for the market category, the annual proportion is assigned to the missing quarter.

$$X_{gps} = \frac{\sum_q \sum_n W_{gpsqn}}{\sum_s \sum_q \sum_n W_{gpsqn}}$$

There is no borrowing across gear groups, port groups or years. That is, if no SpComp sample is taken for a market category by a gear at a port during a specific year, landing for the market category at the port will not be distributed to individual species.

# Biological Data System (BDS)

All biological data are entered directely into the Biological Data System (BDS) by lead technicians/port samplers. The BDS contains information on 106 different species, approximately 44,000 distinct samples (a sample can comprise 1 to over 100 individual fish), and over 1.6 million individual fish records. Commercial groundfish data have been collected since 1954 (starting with Petrale and Pacific cod) and expanded to other species in the 1960's. Research data were added in 1967 (starting with Pacific Ocean Perch) and sport data were added in 1975. More recently the BDS has been used as the repository for non-groundfish species, including but not limited to: Pacific sardine, northern anchovy, hagfish species, etc.

The BDS contains length, weight, sex, maturity, and age data for all groundfish samples. Other information associated with each sample, such as landing date, fish ticket number, gear type, and catch area is also collected. This information is used to update tables in PacFIN and RecFIN, and is used by state and federal managers to set regulations, harvest quotas, and stock assessments. Detailed description of the BDS and data transmission process to PacFIN is reported in Appendix A and Appendix B.

# References

Clark, W. G. 1986. Standardization of Washington's historical trawl logbook data -- progress report. State of Washington Department of Fisheries**.** 116 p.

Clark, W. G. 1986. Washington's trawl logbook data, past and present -- progress report. State of Washington Department of Fisheries**.** 68 p.

Crapo, C., B. Paust, J. Babbitt. 1993. Recoveries & Yields from Pacific Fish and Shellfish. Alaska Sea Grant College Program. Marine Advisory Bulletin 37. 32 p.

Konkel, G. 1998. Operation of the coastal trawl logbook system and Pacific fishery information network reporting. Olympia, WA: Washington Department of Fish and Wildlife. Rep. MRD 98-02. 65 p.

Tagart, V.J. 1997. Groundfish data collection in Washington. In Sampson, D.B., and P.R. Crone. 1997. Commercial fisheries data collection procedures for U.S. Pacific coast groundfish. NOAA Tech. Memo. NMFS-NWFSC-31. 189 p.

Tsou, T.S., L.L. Wargo, M. Langness, C. Jones, R. LeGoff, D. Downs, V. Okimura, B. Walker, D. Bacon. 2015. Washington coastal commercial groudfish port sampling. Olympia, WA: Washington Department of Fish and Wildlife. FPT 15-XX. [W1]44 p.

**Table 1. Conversion factors for groundfish landed in different conditions.**

| Species Name | Dress Condition | Conversion Factor |
|---|---|---|
| PACIFIC HALIBUT | Dressed-Head off | 1.33 |
| | Fillets | 2.041 |
| | Dressed-Head on | 1.17 |
| SOLE-GENERAL | Dressed-Head off | 1.35 |
| | Fillets | 3 |
| SOLE BUTTER | Dressed-Head off | 1.35 |
| | Fillets | 3 |
| SOLE C-O | Dressed-Head off | 1.35 |
| | Fillets | 3 |
| SOLE DOVER | Dressed-Head off | 1.3905 |
| | Fillets | 3 |
| SOLE ENGLISH | Dressed-Head off | 1.35 |
| | Fillets | 3 |
| SOLE PETRALE | Dressed-Head off | 1.35 |
| | Fillets | 3 |
| SOLE REX | Dressed-Head off | 1.3905 |
| | Fillets | 3 |
| SOLE ROCK | Dressed-Head off | 1.35 |
| | Fillets | 3 |
| SOLE SAND | Dressed-Head off | 1.35 |
| | Fillets | 3 |
| SAND DABS | Dressed-Head off | 1.35 |
| | Fillets | 3 |
| STARRY FLOUNDER | Dressed-Head off | 1.35 |
| | Fillets | 3 |
| SABLEFISH | Dressed-Head off | 1.6 |
| | Fillets | 3 |
| | Dressed-Head on | 1.12 |
| LINGCOD | Dressed-Head off | 1.5 |
| | Fillets | 3 |
| | Dressed-Head on | 1.1 |
| PACIFIC COD | Dressed-Head off | 1.75 |
| | Fillets | 3 |
| | Dressed-Head on | 1.23 |

| Species Name | Dress Condition | Conversion Factor |
|---|---|---|
| WALLEYE POLLOCK | Dressed-Head off | 1.75 |
| | Fillets | 3 |
| ROCKFISH (all species) | Dressed-Head off | 1.43 |
| | Fillets | 3 |
| | Dressed-Head on | 1.14 |
| SCULPIN | Dressed-Head off | 1.75 |
| | Fillets | 3 |
| BLUE SHARK | Dressed-Head off | 1.49 |
| | Fillets | 2.5 |
| | Fins | 16.67 |
| SPINY DOGFISH | Dressed-Head off | 1.82 |
| | Dressed-Head on | 1.33 |
| | Fins | 25 |
| MUD SHARK | Dressed-Head off | 1.82 |
| | Fillets | 2.86 |
| | Fins | 20 |
| SOUPFIN SHARK | Dressed-Head off | 1.96 |
| | Fins | 25 |
| BASKING SHARK | Dressed-Head off | 1.72 |
| | Fillets | 3.13 |
| | Fins | 20 |
| THRESHER SHARK | Dressed-Head off | 1.41 |
| | Fillets | 2.27 |
| | Fins | 7.14 |
| SKATE | Dressed-Head off | 3 |
| | Fillets | 3 |
| LONGNOSE SKATE | Dressed-Head off | 3 |
| | Fillets | 3 |

**Table 2. List of logbook transaction types.**

| Transaction Type | Transaction Description |
|---|---|
| 1 | Trip Data |
| 2 | Tow Data |
| 3 | Catch Data |
| 4 | Logbook and Related Fish Ticket |

**Table 3. WDFW Area to PMFC Area conversion.**

| WDFW Area* | PMFC Area |
|---|---|
| 1 | 5C |
| 2 | 5B |
| 3 | 5A |
| 4 | 3D |
| 5 | 3D |
| 6 | 3D |
| 7 | 3C |
| 8 | 3C |
| 9 | 3C |
| 10 | 3C |
| 11 | 3C |
| 12 | 3B |
| 13 | 3C |
| 14 | 3B |
| 15 | 3B |
| 16 | 3A |
| 17 | 3A |
| 20 | 4A |
| 21 | 4A |
| 22 | 4A |
| 23 | 4A |
| 24 | 4A |
| 25 | 4A |

| WDFW Area* | PMFC Area |
|---|---|
| 26 | 4A |
| 27 | 4A |
| 28 | 4A |
| 29 | 4A |
| 30 | 3A |
| 31 | 5E |
| 32 | 2C |
| 33 | 6A |
| 34 | 2B |
| 36 | 2A |
| 83 | 4A |
| 55 | 6A |
| 38 | 1C |
| 40 | 1B |
| 42 | 1A |
| 48 | 8A |
| 49 | 8B |
| 50 | 8C |
| 51 | 7C |
| 52 | 7B |
| 53 | 7A |
| 54 | 6B |
| 99 | 5D |

**Table 4. Ports within the four main port groups: Westport (WES), Bellingham (BEL), Neah Bay (NB), and others (OTH). Associated PacFIN code is also listed.**

| Description | PacFIN Port Code | Species Comp Port Group |
|---|---|---|
| Aberdeen | GRH | WES |
| Alaska (all ports) | AAL | OTH |
| Anacortes | ANA | BEL |
| Bay Center | WLB | WES |
| Bellingham | BLL | BEL |
| Blaine | BLN | BEL |
| Bremerton | OSP | BEL |
| Brinnon | OWA | BEL |
| Copalis Beach | CPL | WES |
| California (all ports) | ACA | OTH |
| Chinook | LWC | WES |
| Coupeville | ONP | BEL |
| Everett | EVR | BEL |
| Friday Harbor | FRI | BEL |
| Hoquiam | GRH | WES |
| Ilwaco | LWC | WES |
| Kelso | OCR | WES |
| Long Beach | WLB | WES |
| La Push | LAP | N-B |
| La Conner | LAC | BEL |
| Neah Bay | NEA | NB |
| Naselle | WLB | WES |
| Olympia | OLY | BEL |
| Oregon (all ports) | AOR | OTH |
| Port Angeles | PAG | BEL |
| Point Roberts | ONP | BEL |
| Port Townsend | TNS | BEL |
| Poulsbo | OSP | BEL |
| Raymond | WLB | WES |
| South Bend | WLB | WES |
| Seattle | SEA | BEL |
| Sequim | SEQ | BEL |
| Shelton | SHL | BEL |
| Tacoma | TAC | BEL |
| Taholah | OWC | WES |

| Description | PacFIN Port Code | Species Comp Port Group |
|---|---|---|
| Tokeland | WLB | WES |
| Vancouver (Wash.) | OCR | WES |
| Westport | WPT | WES |

**Table 5. Sampling strata for rockfish species composition samples.**

| Time Interval | Port Group | Gear Group | Market Category |
|---|---|---|---|
| Quarter 1 | Neah Bay | Trawl | Shelf Rockfish |
| Quarter 2 | Westport | Hook and Line | Slope Rockfish |
| Quarter 3 | Bellingham | | Near Shore Rockfish |
| Quarter 4 | Other | | Shortspine Thornyhead |
| | | | Longspine Thornyhead |
| | | | Darkblotched Rockfish |
| | | | Black Rockfish* |
| | | | Canary Rockfish* |
| | | | Pacific Ocean Perch* |
| | | | Widow Rockfish* |
| | | | Yellowtail Rockfish* |
| | | | Yelloweye Rockfish* |

*These market categories are not currently sampled because there are no longer sorting accuracy issues for these species.

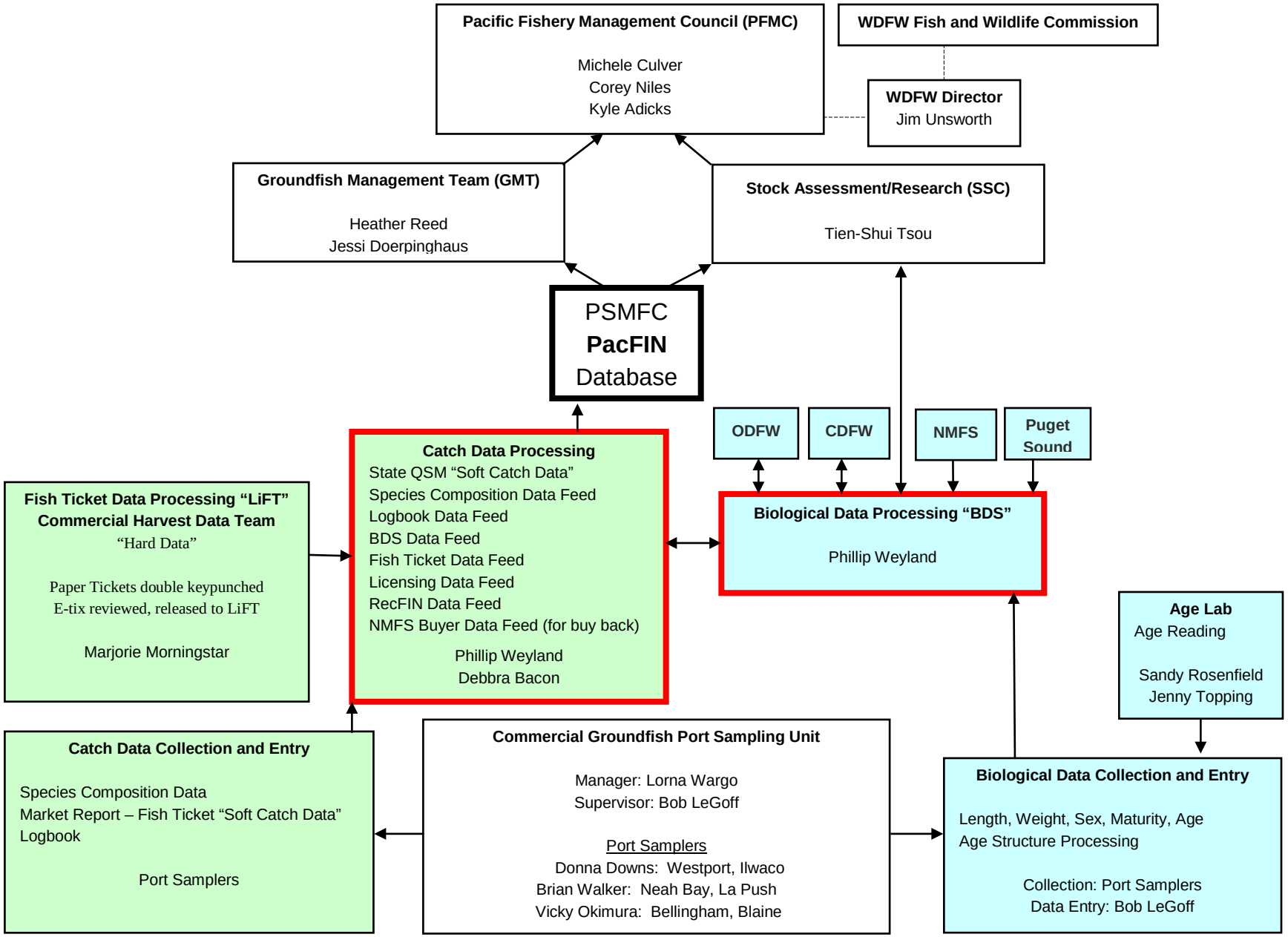
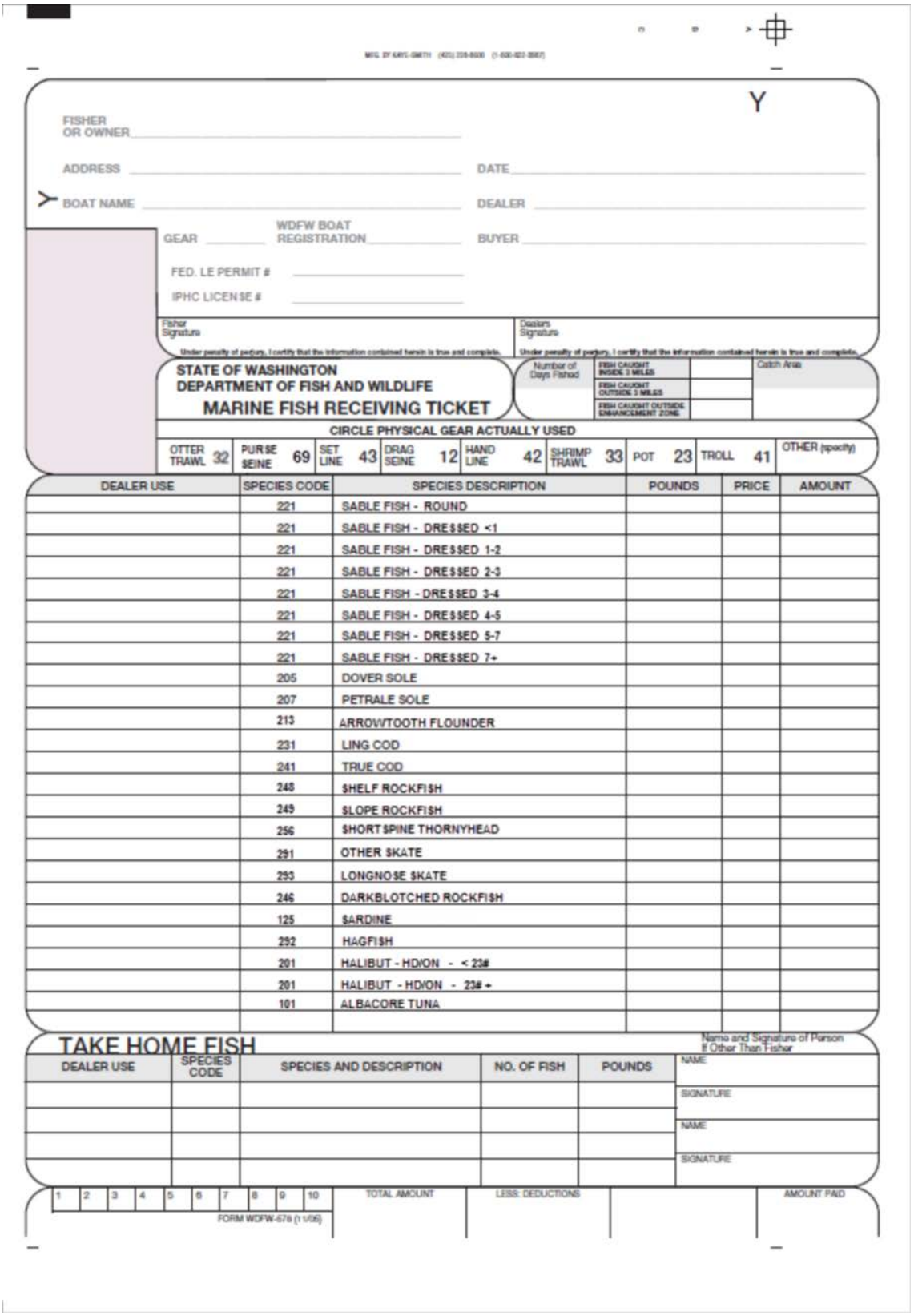



**Figure 1. Systematic flow of commercial groundfish data. Blue = Biological data collection and processing. Green = Catch data collection and processing. Red Boxes = Groups discussed in this report.**



FISHER OR OWNER

ADDRESS

DATE

BOAT NAME

DEALER

GEAR

WDFW BOAT REGISTRATION

BUYER

FED. LE PERMIT #

IPHC LICEN$E #

Fishor Signature

Under penalty of perjury, I certify that the information contained herein is true and complete.

Dealers Signature

Under penalty of perjury, I certify that the information contained herein is true and complete.

**STATE OF WASHINGTON**
**DEPARTMENT OF FISH AND WILDLIFE**
**MARINE FISH RECEIVING TICKET**

Number of Days Fished

FISH CAUGHT INSIDE 3 MILES

FISH CAUGHT OUTSIDE 3 MILES

FISH CAUGHT OUTSIDE ENHANCEMENT ZONE

Catch Area

CIRCLE PHYSICAL GEAR ACTUALLY USED

| OTTER TRAWL 32 | PURSE SEINE 69 | SET LINE 43 | DRAG SEINE 12 | HAND LINE 42 | SHRIMP TRAWL 33 | POT 23 | TROLL 41 | OTHER (specify) |
|---|---|---|---|---|---|---|---|---|

| DEALER USE | SPECIES CODE | SPECIES DESCRIPTION | POUNDS | PRICE | AMOUNT |
|---|---|---|---|---|---|
| | 221 | SABLE FISH - ROUND | | | |
| | 221 | SABLE FISH - DRE$$ED <1 | | | |
| | 221 | SABLE FISH - DRE$$ED 1-2 | | | |
| | 221 | SABLE FISH - DRE$$ED 2-3 | | | |
| | 221 | SABLE FISH - DRE$$ED 3-4 | | | |
| | 221 | SABLE FISH - DRE$$ED 4-5 | | | |
| | 221 | SABLE FISH - DRE$$ED 5-7 | | | |
| | 221 | SABLE FISH - DRE$$ED 7+ | | | |
| | 205 | DOVER SOLE | | | |
| | 207 | PETRALE SOLE | | | |
| | 213 | ARROWTOOTH FLOUNDER | | | |
| | 231 | LING COD | | | |
| | 241 | TRUE COD | | | |
| | 248 | $HELF ROCKFI$H | | | |
| | 249 | $LOPE ROCKFI$H | | | |
| | 256 | $HORT $PINE THORNYHEAD | | | |
| | 291 | OTHER $KATE | | | |
| | 293 | LONGNO$E $KATE | | | |
| | 246 | DARKBLOTCHED ROCKFI$H | | | |
| | 125 | $ARDINE | | | |
| | 252 | HAGFI$H | | | |
| | 201 | HALIBUT - HD/ON - < 23# | | | |
| | 201 | HALIBUT - HD/ON - 23# + | | | |
| | 101 | ALBACORE TUNA | | | |
| | | | | | |

**TAKE HOME FI$H**

Name and Signature of Person If Other Than Fishor

| DEALER USE | SPECIES CODE | SPECIES AND DESCRIPTION | NO. OF FISH | POUNDS |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

NAME

SIGNATURE

NAME

SIGNATURE

1 2 3 4 5 6 7 8 9 10

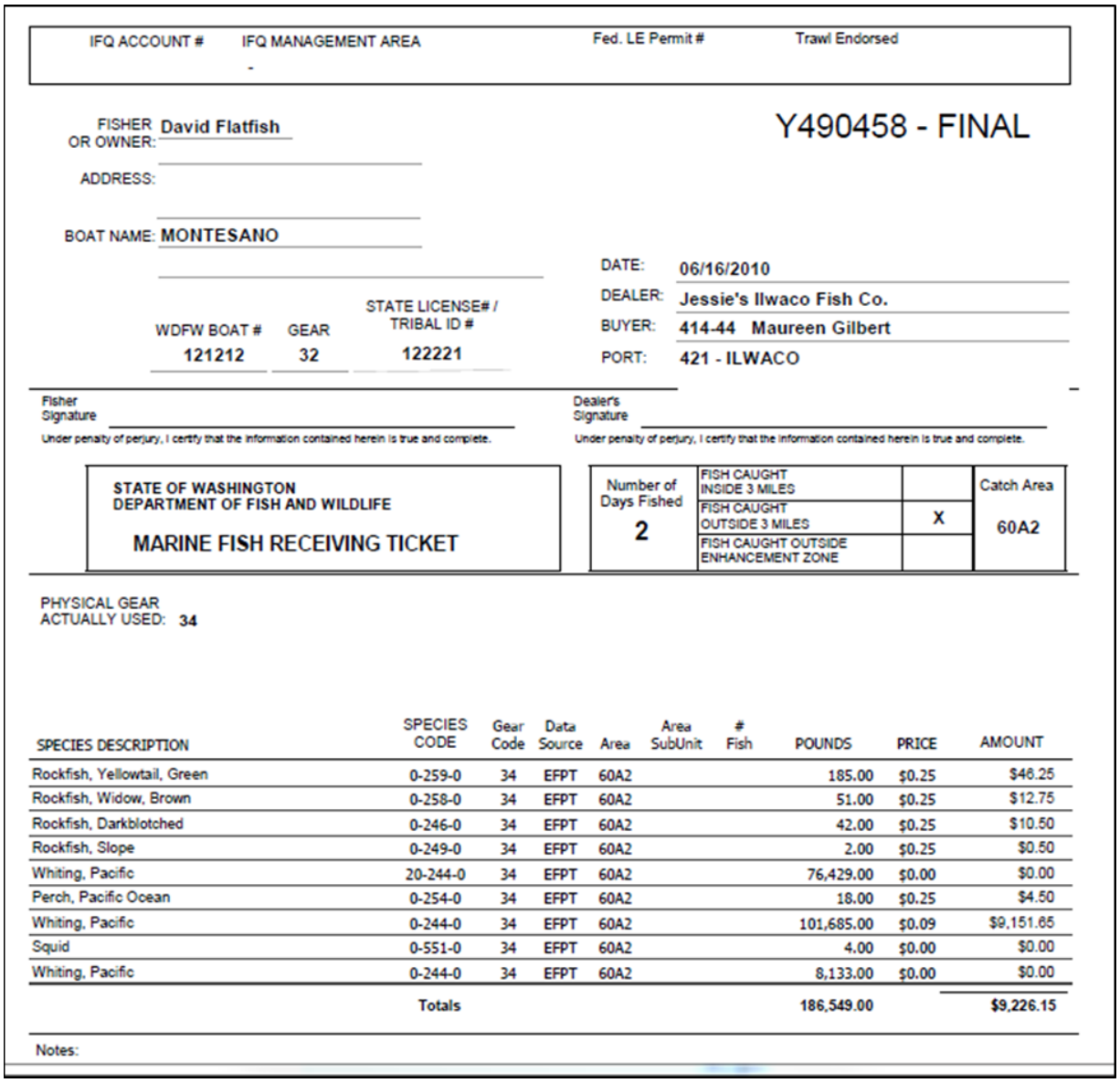
FORM WDFW-678 (1 1/06)

TOTAL AMOUNT

LESS: DEDUCTIONS

AMOUNT PAID

**Figure 2. WDFW paper fish receiving ticket.**



IFQ ACCOUNT # | IFQ MANAGEMENT AREA | Fed. LE Permit # | Trawl Endorsed

-

FISHER OR OWNER: David Flatfish

ADDRESS:

BOAT NAME: MONTESANO

Y490458 - FINAL

DATE: 06/16/2010

DEALER: Jessie's Ilwaco Fish Co.

BUYER: 414-44 Maureen Gilbert

PORT: 421 - ILWACO

| WDFW BOAT # | GEAR | STATE LICENSE# / TRIBAL ID # |
|---|---|---|
| 121212 | 32 | 122221 |

Fisher Signature

Under penalty of perjury, I certify that the information contained herein is true and complete.

Dealer's Signature

Under penalty of perjury, I certify that the information contained herein is true and complete.

STATE OF WASHINGTON
DEPARTMENT OF FISH AND WILDLIFE

MARINE FISH RECEIVING TICKET

| Number of Days Fished | | | Catch Area |
|---|---|---|---|
| 2 | FISH CAUGHT INSIDE 3 MILES | | 60A2 |
| | FISH CAUGHT OUTSIDE 3 MILES | X | |
| | FISH CAUGHT OUTSIDE ENHANCEMENT ZONE | | |

PHYSICAL GEAR ACTUALLY USED: 34

| SPECIES DESCRIPTION | SPECIES CODE | Gear Code | Data Source | Area | Area SubUnit | # Fish | POUNDS | PRICE | AMOUNT |
|---|---|---|---|---|---|---|---|---|---|
| Rockfish, Yellowtail, Green | 0-259-0 | 34 | EFPT | 60A2 | | | 185.00 | $0.25 | $46.25 |
| Rockfish, Widow, Brown | 0-258-0 | 34 | EFPT | 60A2 | | | 51.00 | $0.25 | $12.75 |
| Rockfish, Darkblotched | 0-246-0 | 34 | EFPT | 60A2 | | | 42.00 | $0.25 | $10.50 |
| Rockfish, Slope | 0-249-0 | 34 | EFPT | 60A2 | | | 2.00 | $0.25 | $0.50 |
| Whiting, Pacific | 20-244-0 | 34 | EFPT | 60A2 | | | 76,429.00 | $0.00 | $0.00 |
| Perch, Pacific Ocean | 0-254-0 | 34 | EFPT | 60A2 | | | 18.00 | $0.25 | $4.50 |
| Whiting, Pacific | 0-244-0 | 34 | EFPT | 60A2 | | | 101,685.00 | $0.09 | $9,151.65 |
| Squid | 0-551-0 | 34 | EFPT | 60A2 | | | 4.00 | $0.00 | $0.00 |
| Whiting, Pacific | 0-244-0 | 34 | EFPT | 60A2 | | | 8,133.00 | $0.00 | $0.00 |
| | Totals | | | | | | 186,549.00 | | $9,226.15 |

Notes:

**Figure 3. WDFW electronic (groundfish IFQ) fish receiving ticket.**

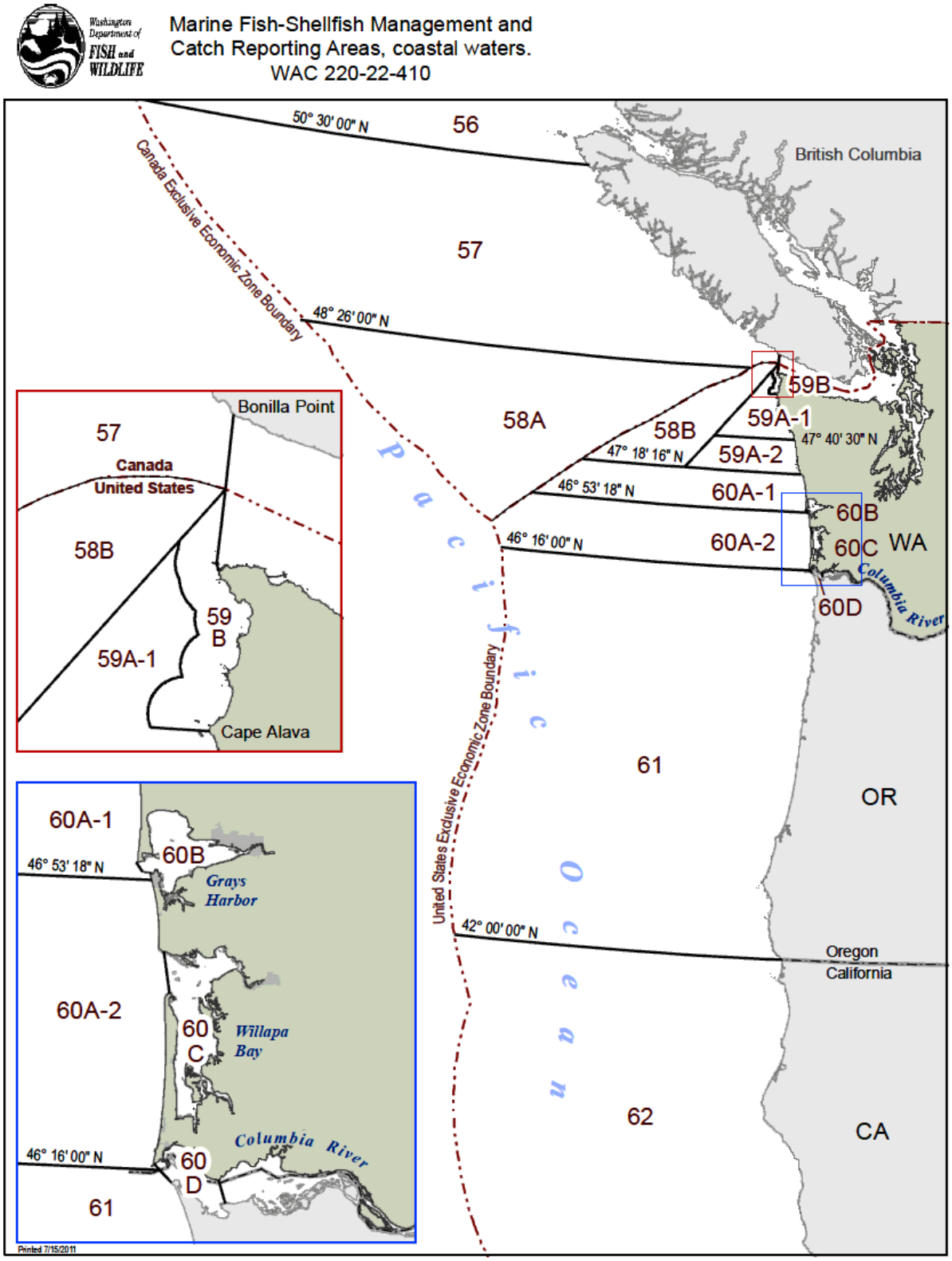



**Figure 4. WDFW marine fish-shellfish management and catch reporting areas.**

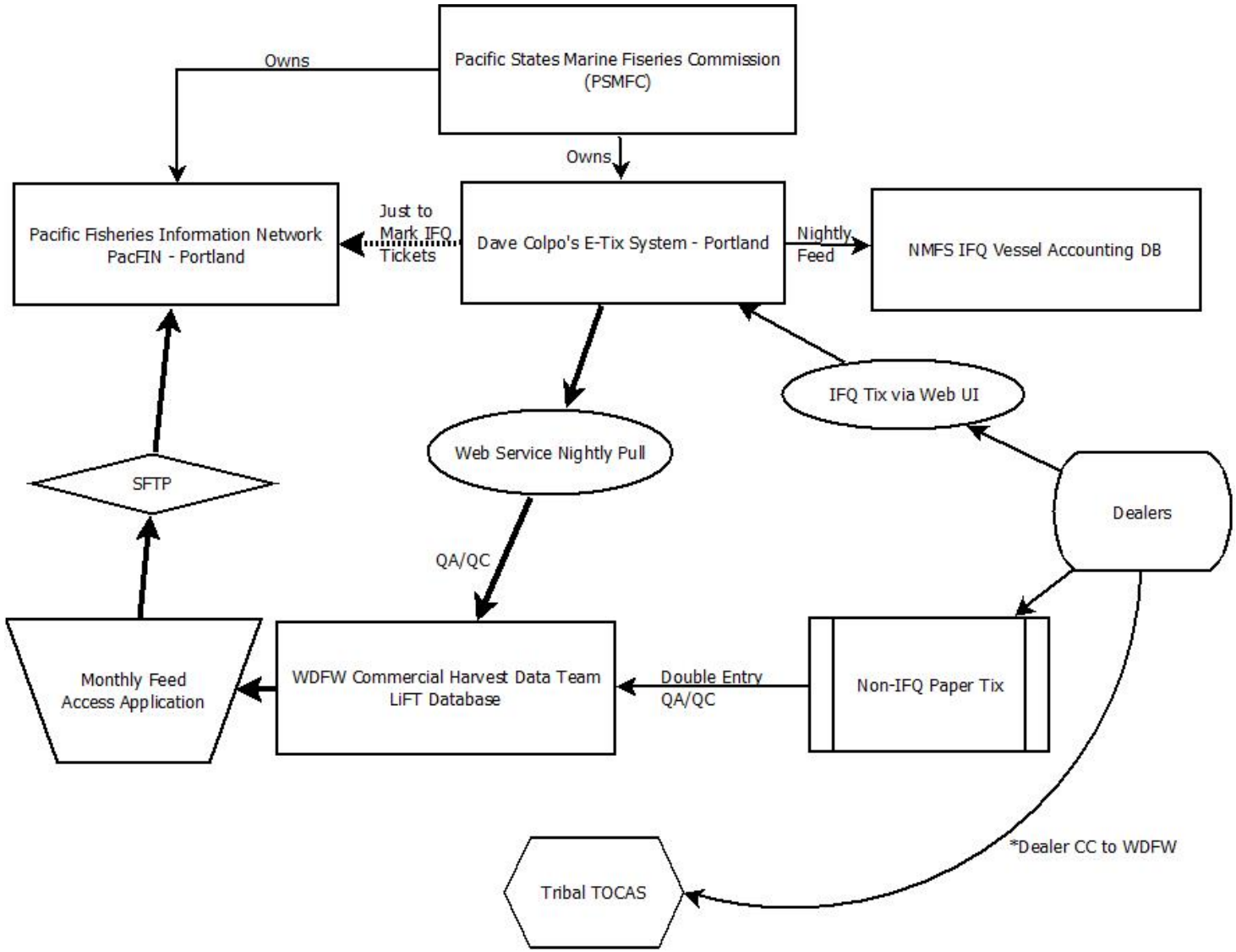



**Figure 5. Fish ticket data flow.**

Vessel Name EXAMPLE Departure: Date 7 (Month) 6 (Day) 10 (Year) Time 0400 (Local – 24-hour) Port WESTPORT, WA

Federal Document No 12345 Return: Date 7 (Month) 8 (Day) 10 (Year) Time 0600 (Local – 24-hour) Port WESTPORT, WA

Crew Size (including Captain) 3

EFP trip (check if yes) X Observed trip (check if yes) X Buyer(s) GENERIC SEAFOODS

| DATE mo/day | | TIME Local 24-hour clock | LATITUDE Degrees | LATITUDE Minutes | LONGITUDE Degrees | LONGITUDE Minutes | Average depth of catch (fathoms) | NET TYPE | Target Strategy | Estimated pounds retained each tow – enter 3 or 4-letter code from species code list: SABL | DOVR | LSPN | SSPN | WDOW | YTRK |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7/6 | set | 1300 | 47 | 58•7 | 125 | 47•3 | 500 | L | DTS | 300 | 4,000 | 500 | 100 | | |
| | up | 1730 | 48 | 02•6 | 125 | 45•5 | | | | | | | | | |
| 7/7 | set | 0800 | 47 | 20•3 | 125 | 28•3 | 575 | L | DTS | 100 | 5,000 | 800 | 150 | | |
| | up | 1400 | 47 | 46•4 | 125 | 34•4 | | | | | | | | | |
| 7/7 | set | 1800 | 46 | 52•6 | 124 | 53•2 | 90 | D | WDOW | | | | | 16,000 | 500 |
| | up | 2200 | 46 | 54•1 | 124 | 53•6 | | | | | | | | | |
| | set | | | • | | • | | | | | | | | | |
| | up | | | • | | • | | | | | | | | | |
| | set | | | • | | • | | | | | | | | | |
| | up | | | • | | • | | | | | | | | | |
| | set | | | • | | • | | | | | | | | | |
| | up | | | • | | • | | | | | | | | | |
| | set | | | • | | • | | | | | | | | | |
| | up | | | • | | • | | | | | | | | | |
| | set | | | • | | • | | | | | | | | | |
| | up | | | • | | • | | | | | | | | | |
| | set | | | • | | • | | | | | | | | | |
| | up | | | • | | • | | | | | | | | | |

REMARKS:

Signed: John Doe

TO BE COMPLETED BY AGENCY

| VESSEL | FISH RECEIVING TICKET NO. |
|---|---|
| PORT | |

**Figure 6. Trawl gear logbook example.**

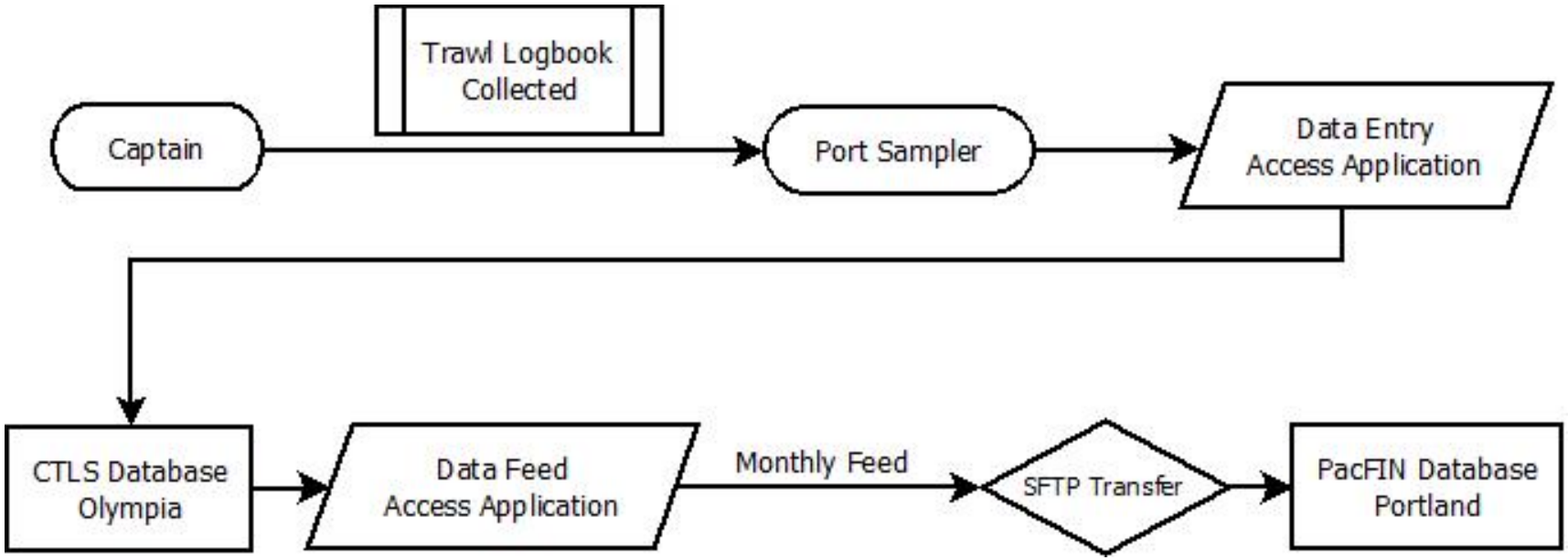



**Figure 7. Trawl logbook data flow.**

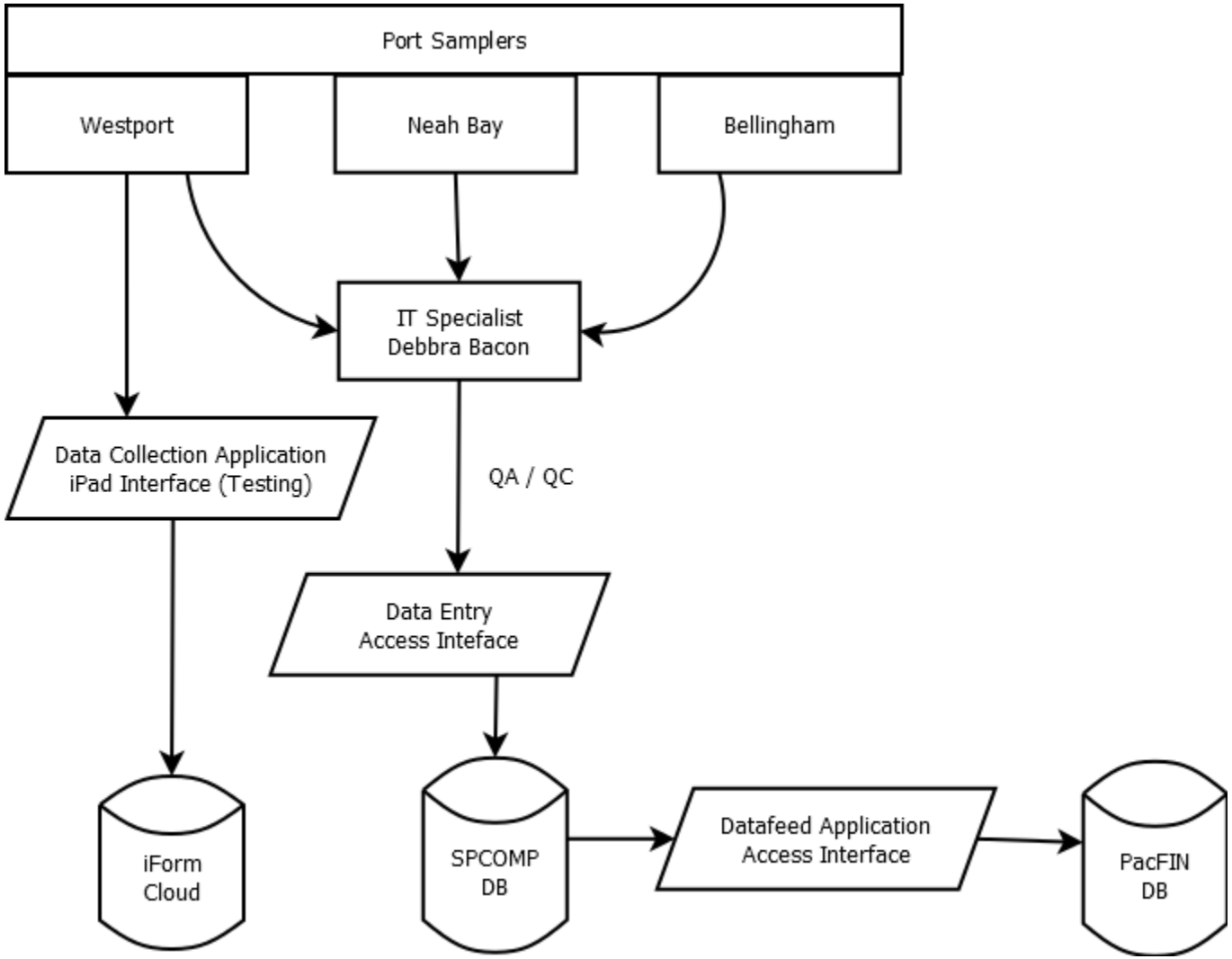



**Figure 8. Species composition data flow.**

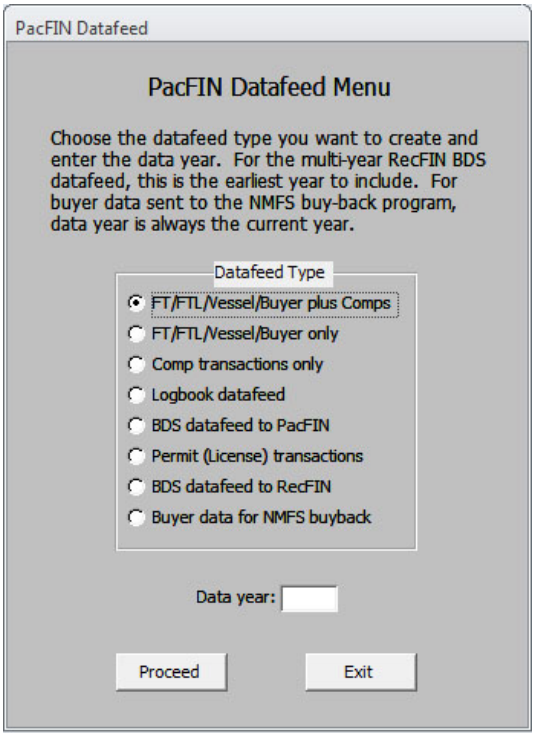
# Appendices

## Appendix A: Documentation for PacFIN/RecFIN Datafeed Process

This document is intended to accompany the process flow diagram (Appendix U) for the PacFIN/RecFIN datafeed providing more detail about objects and process steps depicted in the diagram. The diagram and this document both provide system documentation for the consolidated, menu-driven PacFIN/RecFIN datafeed process that is entirely controlled by the Microsoft Access database application *datafeed.mdb*. This document is organized to follow the sequence of steps in the process flow diagram, grouping objects by the major sub-processes (i.e., individual datafeeds). For readability, code excerpts such as SQL queries or VBA scripts are not included in this document. Anyone familiar with Access can open these database objects in design mode and examine them. Access object names—tables, queries, etc.—are in boldface so they can be spotted easily in the document.

Main Menu Form

The main menu form, **frmStartUp**, is launched automatically when the Access database is opened. The user interface looks like this:




This is the only user interaction required to run all of the datafeed types listed on the form. However, some datafeeds require follow-up action by the user, outside of Access, once the

datafeed is created. Besides running the processes necessary to produce the selected datafeed, the form does some housekeeping on exit, such as emptying scratch tables that need to be retained (to protect queries from being broken if opened in design mode) and deleting temporary tables. The form stays alive until the Exit button is pressed, so more than one datafeed can be run without reopening the Access application.

There is one other action taken by the form and that is to populate the small local table **DatafeedYear**. This table has only two fields and one row, the field *Year* stores the selected datafeed year and the field *Type* stores a short abbreviation for the datafeed type (e.g., "lbk" for logbook datafeed). The datafeed year value is used in many instances by datafeed sub-processes. The datafeed type abbreviation becomes part of the final delivered filename for all PacFIN datafeeds. Here is the naming convention for all PacFIN datafeed files: *xxxyyyy.yymmdd*, where *xxx* is the datafeed type abbreviation, *yyyy* is datafeed year, and *yymmdd* is the date the datafeed file was created. Here are the PacFIN datafeed type abbreviations: *ft* = fish ticket (including vessel and buyer transactions) without species and area comps, *ftc* = fish ticket with comps, *cmp* = comp transactions only, *lbk* = Coastal Trawl Logbook (CTL), *bds* = Biologic Data System (BDS), and *pmt* = permit transactions. The output file naming is different for the BDS RecFIN and NMFS buyback datafeeds, as will be explained later in the sections on those sub-processes.

Fish Ticket/Vessel/Buyer Datafeed (sub-process 1)

The actions performed by the Access application to produce the fish ticket datafeed are pretty simple. Most of the heavy lifting is done by a set of SQL stored procedures that reside with the Licensing and Fish Ticket System (**LiFT 2000**). I will discuss the actions performed locally by the Access application, and then address the SQL stored procedures. Since these stored procedures are not embedded in the Access application as are other objects discussed in this document, the SQL code for them is provided in Appendix I.

The macro **MakeFTDatafeed** controls the fish ticket datafeed creation process. The first thing it does is run query **qryAddFTHeader**, which inserts a PacFIN header row at the beginning of the datafeed file. These header rows are standard for all PacFIN datafeeds, telling the loader program what type of datafeed it is and providing an identification string for the results report. The next thing it does is launch the VBA module **vbaRunDatafeedSP**, which is a very short VBA script that: 1) reads the selected datafeed year from the table **DatafeedYear**, and 2) executes the parent stored procedure for the fish ticket/vessel/buyer datafeed process in **LiFT 2000**. The command string that is passed by the VBA module to **LiFT 2000** looks like this: exec spa_pacfin_datafeed 2012. This command string is run as a SQL pass-through query that uses a database connection string embedded in the VBA module. The fully formatted datafeed is dumped into a scratch table called **pacfin_datafeed** in **LiFT 2000**, which is represented as an ODBC table link by the same name in Access.

The next thing the macro does is run the query **qryGetFTDatafeed**, which simply downloads the fully formatted datafeed from **LiFT 2000** and dumps it into a local table called **DatafeedOut**. Finally, the macro runs the query **qryFixFTNumShift**, which fixes the problem of the fish ticket numbers being shifted one character to the right in some fish ticket transactions. This problem could not be corrected in the stored procedure because it's sporadic, inconsistent and has no explanation in the source data or the programming. Upon completion of the macro, a fully formatted fish ticket/vessel/buyer datafeed resides in local table **DatafeedOut**.

The following stored procedures are executed on the **LiFT 2000** server when the fish ticket datafeed is created:
**spa_pacfin_datafeed** – The parent stored procedure that executes the others. It requires a datafeed year argument.
**spa_pacfin_fishticket_feed** – Creates the fish ticket portion of the datafeed. Requires a datafeed year argument that is passed by **spa_pacfin_datafeed**.
**spa_pacfin_vessel_feed** – Creates the vessel portion of the datafeed. Requires a datafeed year argument that is passed by **spa_pacfin_datafeed**.
**spa_pacfin_buyer_feed** – Creates the buyer portion of the datafeed. Requires a datafeed year argument that is passed by **spa_pacfin_datafeed**.

Appendix I contains the SQL code for these stored procedures. All of them contain extensive in-line comments (lines beginning with "--") that should be adequate as documentation.

Comp Transactions (sub-process 2)

This is by far the most complex sub-process in the Access application, mainly due to the many calculations and data manipulations required to create rockfish species comp, area comp, and aggregated effort transactions. This sub-process would be a good candidate for replacement by VBA modules, which would use array variables and matrix math rather than queries and temporary tables. However, the logic for these VBA modules would still look something like the existing sub-process diagram. Due to its complexity and duplication of some steps, sub-process 2 in the diagram has three additional sub-processes, 2a, 2b, and 2c.

Comp transactions can either be appended to a fish ticket/vessel/buyer datafeed or delivered as a stand-alone datafeed. In the latter case, it's necessary to include a PacFIN header row at the beginning of the datafeed file. This is done by the query **qryAddCompHeader**. Since the fish ticket/vessel/buyer datafeed already has a header row, this step is unnecessary when comp transactions are appended to it.

The macro **MakeCompMaster** controls the entire comp datafeed process, including the creation of all three comp transaction types. The first step in the macro is to run a pass-through SQL query called **qryFillMissingData**. This query executes a SQL stored procedures called **fill_missing_data** that resides in the SQL Server database **MFCTL**. It fills derived values in the

main CTL data tables, such as catch area (based on lat/lon), tow time, number of tows, days out, and days fished. **qryFillMissingData** also executes a similar stored procedure called **fill_spcomp_data**, which fills derived values in the species comp dataset that co-exists in **MFCTL**, such as total N, total weight, and percent by weight. For accuracy, consistency, and to save time, these values are not entered by the user, but are filled by these stored procedures as a scheduled batch process during non-work hours. However, these values are essential for the datafeed, so the stored procedures must be run at the beginning of the comp datafeed process.

The next step in the macro is to run the query **qryCopyPACFIN_LBK**, which along with its sub-query **qrySelectPACFIN_LBK**, downloads logbook data from the PacFIN Oracle database and copies it to the local table **PACFIN_LBK_TMP**. The SQL pass-through query **qrySelectPACFIN_LBK** selects a subset of WDFW data for year 2005 and later from PacFIN tables: **lbk_trip**, **lbk_tow**, and **lbk_catch**. The purpose of this download is to obtain fish ticket-adjusted logbook data from PacFIN for use in calculating area comps. WDFW no longer does fish ticket-logbook reconciliation locally, as this is now handled on the PacFIN side.

The purpose of rockfish species comp (SCM) transactions is to allow landings for rockfish market categories that include multiple species, such as the nearshore rockfish assemblage, to be broken out into individual rockfish species. Rockfish species comps are the first to be calculated by the **MakeCompMaster** macro and two sets of comps are created, one for trawl gear (TWL) and one for hook-and-line gear (HKL). Since the process for calculating these comps is essentially the same, a sub-macro, **MakeSppCompSub**, is called twice from **MakeCompMaster**, once for each gear type. As the name implies, **MakeSppCompSub** manages all the calculations and data manipulations needed to create one set of species comps. The actions of **MakeSppCompSub** are depicted as sub-process 2a in the diagram. However, before running this macro, a simple step is taken to select the desired gear type. The query **qrySelectTWL** writes the string "TWL" into the small local table **SpCompGear**, which has only one row and one field called *GearClass*. Sub-process 2a uses the *GearClass* value six times during the process of creating species comps. Likewise, the query **qrySelectHKL** writes the string "HKL" into the table **SpCompGear** just before **MakeSppCompSub** is run for the second time.

The query **qryCalcProportionsSC** uses three sub-queries to create species comp proportions from existing species comp data. Sub-query **qrySumWgtByMktCat** sums weights by calendar quarter, port group, and market category for the selected gear type. Sub-query **qrySumWgtBySpecies** sums weights by quarter, port group, market category, and sampled rockfish species for the selected gear type. Sub-query **qryCountySample** calculates N (sample count) by quarter, port group, and market category for the selected gear type. **qryCalcProportionsSC** then calculates proportion by dividing the results from **qrySumWgtBySpecies** by those from **qrySumWgtByMktCat** and then combines them with the output from **qryCountySample** to produce a complete set of species comps, which are stored in local table **NewComps**. **qryCalcProportionsSC** gets the datafeed year from the table **DatafeedYear** and uses the **PORT**

table in **MFCTL** to expand WDFW port groups into PacFIN ports. In summary, the steps involved in creating species comps are:

1. Sum weights by calendar quarter, port group, and market category for the selected gear
2. Sum weights by quarter, port group, market category, and rockfish species for the selected gear
3. Divide results from step 2 by those from step 1 to get species proportions by quarter, port group, and market category for the selected gear
4. Count samples by quarter, port group, and market category for the selected gear
5. Combine the results from step 3 with those from step 4 to create species comp transactions

The mathematical formula for calculating species comp proportions can be found in Appendix J. The query **qryCalcAnnualNUS** works in a similar fashion to **qryCalcProportionsSC**, except that proportions are computed for the whole year instead of by quarter and only for market categories that begin with "NUS" (i.e., the three unspecified rockfish categories, nearshore, slope, and shelf). This query uses sub-queries **qrySumWgtByMktCatNUS** and **qrySumWgtBySpeciesNUS** in a similar fashion to their counterparts in the previous paragraph and both obtain the datafeed year from **DatafeedYear**. However port groups are not expanded into PacFIN ports (that comes later in the "borrowing" process explained in the next paragraph). The results of **qryCalcAnnualNUS** are stored in local table **AnnualCompsNUS**.

The purpose of query **qryAddDefaultNUS** is to find empty cells in the matrix of species comps for the "NUS" market categories and then fill them using the annualized comps stored in **AnnualCompsNUS**. Sub-query **qryNeedsNUScomps** determines where the empty cells are using sub-queries **qryAllPossibleNUS** and **qryHasNUScomps**. **qryAllPossibleNUS** uses the table **AnnualCompsNUS** and a small table **Quarters** (which contains the list of quarters 1, 2, 3, and 4) to determine what the full NUS species comp matrix should look like. **qryHasNUScomps** determines which cells in the matrix are filled with real comps. Then top-level query **qryAddDefaultNUS** adds default comps for the empty cells to table **NewsComps** using the information passed from **qryNeedsNUScomps**. It also expands WDFW port groups into PacFIN ports using the **PORT** table. This "borrowing" process uses annualized comps to fill chronological gaps (empty quarters) for the same port group and market category. There is no borrowing between port groups or market categories.

The two queries **qryAdjustedProportions** and **qryMakeAdjustments** have a very simple purpose that can be summarized briefly. Together, they make sure that all species comp proportions for a given quarter, port, and market category add up to 1.0. **qryAdjustedProportions** uses sub-queries **qryProportionsToAdjust** and **qryCalcAdjustments** along with data in the table **NewComps** to create a list of adjustments needed, which is stored in local table **Adjustments**. **qryMakeAdjustments** then applies these adjustments to the appropriate comps in the table **NewComps**. The adjustment process subtracts the sum of all proportions for a given quarter, port, and market category from 1.0, then adds or subtracts the difference from the largest proportion as necessary to force them to add up to 1.0. These small differences are due to rounding in the

calculation of proportions, and this adjustment process is necessary to avoid error messages when the datafeed is loaded into PacFIN.

The next step in the process controlled by macro **MakeSppCompSub** involves filling the species comp matrix with default comps for market categories that are not being sampled. These nearly pure market categories (canary rockfish, widow rockfish, POP, etc.) would remain as nominal categories rather than actual rockfish species if dummy comps are not included in the datafeed. For example, widow rockfish would remain as WDW1, a nominal category, if dummy proportions of 1.0 are not applied to all the possible quarters and ports, thus converting it to specific rockfish code WDOW. There's a local table called **DefaultComps** that contains all the possible quarters and ports for all the "pure" market categories along with a default proportion of 1.0 for each of them. There's also a *Yes/No* field in the table called *HasComps*, which is used to flag any cells in the matrix that actually have comps and thus don't need default proportions. Query **qrySetHasCompsNo** first sets the *HasComps* flag to *No*, and then query **qryFlagHasComps** sets the *HasComps* flag to *Yes* for cells that do have real comps based on data already in table **NewComps**. Finally, query **qryAddDefaultComps** adds the needed default proportions to the table **NewComps** using the information in the table **DefaultComps** (i.e., cells with *HasComps = No*).

The next step in the species comp process is to reformat the data in **NewComps**, an Access table with multiple fields, into the PacFIN transaction format, which is a long, column-sensitive character string. This step is done by query **qryFormatCompOutput** and the resulting transactions are stored in local table **CompsOut**, which has a single, wide character field. The final step in sub-process 2a is to append the species comps in table **CompsOut** to the main datafeed table **DatafeedOut**. This is done by the query **qryAppendCompsToDF**.

After completion of sub-process 2a (macro **MakeSppCompSub**), control is returned to sub-process 2 (macro **MakeCompMaster**). The next step in sub-process 2, after species comps have been created for both TWL and HKL gears, is to produce area comps.

The purpose of area comp (ACM) transactions is to provide a means for distributing landed catch reported on fish tickets into Pacific Marine Fisheries Commission (PMFC) catch areas based on tow locations reported in trawl logbooks. Compared to the species comp creation process, the production of area comps is pretty simple. As with species comps, the calculation of area comps is actually a matrix math operation. Query **qryMakeAreaComps** obtains the necessary data from sub-query **qryCalcProportionsAC** and inserts fully formatted area comp transactions into table **CompsOut**. Sub-query **qryCalcProportionAC** works much like **qryCalcProportionsSC**, using three sub-queries in a similar fashion, all of which pull data from local table **PACFIN_LBK_TMP**. **qrySumWgtByPortSpp** aggregates adjusted catch by month, trip type (Puget Sound or coastal), port, and market category. **qrySumWgtByPortSppArea** aggregates adjusted catch the same as **qrySumWgtByPortSpp**, but with the addition of another dimension, PMFC catch area. **qryCalcCountTrips** produces the N value needed for area comps.

**qryCalcProportionAC**  then divides **qrySumWgtByPortSppArea** results by those from **qrySumWgtByPortSpp** to obtain area proportions and combines them with N values from **qryCalcCountTrips** to produced completed area comps. In summary, the steps involved in creating area comps are:

1. Aggregate adjusted catch by month, trip type, port, and market category
2. Aggregate adjusted catch by month, trip type, port, market category, and catch area
3. Divide results from step 2 by those from step 1 to get area proportions by month, trip type, port, and market category
4. Count trips by month, trip type, port, and market category
5. Combine the results from step 3 with those from step 4 to create area comp transactions

Local tables **PORT_AC** and **SPECIES_LBK** are used to convert WDFW port codes to their PacFIN equivalent and WDFW species codes to corresponding PacFIN codes. As with many other process steps, the datafeed year is obtained from table **DatafeedYear**. As with species comps, query **qryAppendCompsToDF** is used to add the area comps to table **DatafeedOut**. The mathematical formula for calculating area comp proportions can be found in Appendix J.

The purpose of aggregated effort (AGE) transactions is to provide a summary of trawl effort based on the tow durations reported in logbooks. The creation of aggregated effort transactions is even more straightforward. Query **qryMakeAggEffort** obtains the necessary data from sub-query **qryCalcAggEffort** and inserts fully formatted aggregated effort transactions into table **CompsOut**. **qryCalcAggEffort** aggregates tow durations from table **PACFIN_LBK_TMP** by month, catch area, and port. Table **PORT_AC** is used to convert WDFW port codes to corresponding PacFIN port codes. Datafeed year is taken from table **DatafeedYear**. Finally, query **qryAppendCompsToDF** is used to add the aggregated effort transactions to table **DatafeedOut**.

Export Datafeed (sub-process 3)

This small sub-process, controlled by macro **ExportDatafeed**, is used for both fish ticket and comp datafeeds to export the final transactions from Access table **DatafeedOut** to Windows text file **datafeed.txt**. First, the macro runs query **qryRTrimOutput**, which strips trailing blanks from all rows in **DatafeedOut**. Then **ExportDatafeed** uses built-in macro action *ImportExportText* to export the contents of table **DatafeedOut** to Windows file **datafeed.txt**.

Upload Datafeed (sub-process 4)

This sub-process, initiated by macro **UploadDatafeed**, mainly uses a batch file and small utility programs running in the Windows environment to upload the final datafeed file to PacFIN. Although the macro itself can launch a batch file, VBA module **vbaRunBatchFile** is used instead to run **upload_datafeed.bat** because it's able to pass variables to the batch file, whereas a macro cannot. The three text variables passed are the datafeed type abbreviation (2 or 3 characters), datafeed year (*yyyy*), and run date (*yymmdd*). These are used by the batch file to rename

**datafeed.txt** to its final name following the naming convention described previously in the *Main Menu Form* section. Besides renaming the datafeed file, the batch file does the following:

- Uses the small utility **dos2unix.exe** to strip carriage-return characters from the datafeed file leaving only line-feed characters as the Unix-friendly row terminators
- Uses the stand-alone zip utility **7za.exe** to zip the datafeed file into file **datafeed.zip**
- Uses **WinSCP.exe** along with a batch script, **ftp_script.txt**, to upload **datafeed.zip** to the PacFIN "prowfish" server using the *sftp* protocol, unzip the file on "prowfish", and delete **datafeed.zip** on "prowfish"
- Deletes **datafeed.zip** in the local working directory

The end result is a final, delivered datafeed file on the PacFIN "prowfish" server, and another copy left in the local working directory.

CTL Datafeed (sub-process 5)

The CTL datafeed to PacFIN uses a new, streamlined specification that converts from the WDFW relational data model to the very similar PacFIN data model, pulling data from the **TRIP**, **HAUL**, and **CATCH** tables in the SQL Server database **MFCTL**. This process is controlled by the macro **MakeCTLDatafeed.** As with the comp datafeed, the first step is to run **qryFillMissingData** to fill missing values in the main CTL data tables (see explanation in the *Comp Transactions* section). Next, query **qryEmptyCTLDatafeed** empties the output table **CTLDatafeed**. Query **qrySelectCTLDatafeed** is the heart of the datafeed process, using two sub-queries to create a complete CTL datafeed, already partially formatted for delivery to PacFIN. Sub-query **qrySelectCTLTripData** pulls the necessary data from the **TRIP** table for the selected datafeed year. Sub-query **qrySelectCTLCatchData** pulls the catch data for this set of trips from the **CATCH** table. Local table **SPECIES_LBK** converts WDFW species codes to corresponding PacFIN codes. Main query **qrySelectCTLDatafeed** also pulls the necessary data directly from the **HAUL** table, then combines it with data from the two sub-queries and stores it in **CTLDatafeed**. As mentioned, data stored in **CTLDatafeed** is already partially formatted for delivery to PacFIN with three character strings—one for trip data, one for haul data, and one for catch data—but retaining key fields that identify the trip, haul and species.

The last step in the process is to run VBA module **vbaCTLDatafeed**, which does the final formatting for delivery to PacFIN. Rather than read directly from table **CTLDatafeed**, it uses query **qrySortCTLOutput** as an intermediary to make sure the data are sorted correctly. The actions performed by **vbaCTLDatafeed** are:

1. Adds a numeric transaction type code (1 through 4) at the beginning of each row
2. Adds "tow count", a sequential number used by PacFIN to relate haul rows to corresponding catch rows in their data model
3. Separates the list of fish ticket numbers associated with each trip, writing out one per row as transaction type 4

The output from **vbaCTLDatafeed** goes directly into Windows file **datafeed.txt**.

PacFIN BDS Datafeed (sub-process 6)

The BDS datafeed is controlled by macro **MakeBDSdfPacFIN**. The reason "PacFIN" is in the name is because there's another BDS datafeed for RecFIN, controlled by macro **MakeBDSdfRecFIN**. The strategy for creating the PacFIN BDS datafeed is different than previously discussed datafeeds because the output goes into five Access temporary tables that are exported as five separate text files and assembled into a single datafeed file in the Windows environment. The reason for this is that the BDS datafeed specification uses comma-delimited CSV format, which Access macros can output directly using the *ImportExportText* macro action. Using this feature of Access saves a lot of programming to convert tabular data into CSV text strings. All of the BDS tables mentioned below reside in the SQL Server database **MFBDS**.

Query **qryMakeBDSRptRec** creates a header row for the datafeed that goes into temporary table **PF_bds_report**. Query **qryMakeBDSSample** assembles the data needed for the BDS sample transactions, pulling from the BDS **HEADER** table and placing the results in temporary table **PF_bds_sample**. Query **qryMakeBDSCluster** assembles the data needed for the BDS cluster transactions, also pulling from the **HEADER** table and placing the results in temporary table **PF_bds_cluster**. For WDFW, the cluster transactions are actually "dummies" with all fish detail data for a sample included in one cluster, but the PacFIN specification requires them. Query **qryMakeBDSFishDtl** assembles the data needed for the BDS fish detail transactions, pulling from both the BDS **HEADER** and **FISHDTL** tables and placing the results in temporary table **PF_bds_fishdtl**. Next, three very similar queries—**qryMakeBDSAge1**, **qryMakeBDSAge2**, and **qryMakeBDSAge3**—are run one after the other, assembling data needed for the BDS fish age transactions. These queries pull fish ageing attribute information from the **HEADER** table and the actual ages from the **FISHDTL** tables, placing the results in temporary table **PF_bds_age**. The only difference between these three queries is which field in the **FISHDTL** table they pull data from (the WDFW BDS data model has three fish age columns).

After all data needed for the BDS datafeed are assembled into the five temporary tables, a series of five *ImportExportText* macro actions are executed by **MakeBDSdfPacFIN**, exporting the contents of the tables as five comma-delimited text files, **PF_bds_report.txt**, **PF_bds_sample.txt**, **PF_bds_cluster.txt**, **PF_bds_fishdtl.txt**, and **PF_bds_age.txt**. Finally, the Windows batch file **bds_datafeed.bat** is launched by **MakeBDSdfPacFIN** to append the five text files into a single datafeed file called **datafeed.txt** that is ready for uploading to PacFIN.

PacFIN Permit Datafeed (sub-process 6)

This datafeed (a.k.a. "W-O-C" permit) is only sent to PacFIN once or twice a year. It provides a list of "permits" and who owns them. However, for WDFW, "permits" translates into the various marine fish and shellfish commercial fishing license types that can be found on the agency's website (currently, there are more than 40 of them). There are several steps involved in creating permit transactions, but it's mainly just a process of selecting and properly formatting the

necessary data from licensing tables in **LiFT 2000**. Besides the usual header row, there are three transaction types: a list of permit types with attributes, permit owners with their fishing license number and some contact information, and data describing permits in effect during the datafeed year, including the registration number of the assigned vessel. Like the PacFIN BDS datafeed, the permit datafeed specification uses comma-delimited CSV format.

The permit datafeed sub-process is controlled by macro **MakePermitTrans**. Query **qryPermitHeader** creates the standard header transaction and inserts it into local table **tblPermitHeader**. Query **qryPermitAttributesAll** uses sub-query **qryLicenseTypeByYear** along with **LiFT 2000** tables **LICENSE_TYPE** and **LICENSE_TYPE_FEE** to assemble all the data needed for the permit attribute transactions, which it places in local table **tblPermitAttribute**. Actually, the fee information is not needed for permit attribute transactions, but **tblPermitAttribute** is also used as a data source for other queries that does need this information. Finally, query **qryPermitAttributeFinal** adds a transaction type identifier to selected data from **tblPermitAttribute** (leaving off the fee information) and places the result in **tblPermitAttributeFinal**, which is ready for export.

Query **qryPermitOwner** uses sub-query **qryOwnerData** to pull selected information about permit owners from the **LiFT 2000** licensing tables and places it in local table **tblPermitOwner**. Query **qryPermitSMB** uses sub-query **qryLicenseSMB** to pull vessel-related information from the **LiFT 2000** licensing tables and places it in local table **tblPermitSMB**. The "SMB" reference in these names stands for State Marine Board. There's a problem with the creation of the owner and vessel datasets in that the activation and expiration dates don't always match, leaving gaps in the time series. For example, the fishing license may run from January 1 to December 31, but for whatever reason, the associated vessel registration might not be activated until January 12. This leaves a gap at the beginning of the year where a license that requires a vessel has no assigned vessel. There's a series of three process steps that fill these gaps with dummy rows in table **tblPermitSMB** for the missing time periods, but with no assigned vessels. These dummy rows end up in the final permit transaction portion of the datafeed, which prevents problems on the PacFIN side because their loader program checks for these date mismatches. Query **qryEmptySMBGaps** empties the local table **tblPermitSMBGaps**. The VBA module **vbaFillSMBGaps** does the job of creating the necessary dummy rows, pulling data from **tblPermitSMB** and placing the results in **tblPermitSMBGaps**. Finally, query **qryAppendSMBGaps** appends these dummy rows back into **tblPermitSMB**.

Query **qryPermitTransRunLast**, as the name implies, is the last major step in creating the permit transactions, i.e., data describing actual permits. It uses sub-query **qryPermitTransPenult** (second to last) to read from tables **tblPermitOwner** and **tblPermitSMB**, assemble all the necessary data elements and reformat them as necessary, placing the resulting dataset in table **tblPermitTrans**. Next, a series of two queries, **qryEmptyPermitOwnerFinal** and **qryEmptyPermitTransFinal,** are used empty the final output tables, **tblPermitOwnerFinal** and **tblPermitTransFinal**. Then queries **qryPermitOwnerFinal** and **qryPermitTransFinal** do some

minor formatting, add a transaction identifier, and place the results in tables**tblPermitOwnerFinal** and **tblPermitTransFinal**, which are ready for export. Next, a series of four *ImportExportText* macro actions export tables **tblPermitHeader**, **tblPermitAttributeFinal**, **tblPermitOwnerFinal**, and **tblPermitTransFinal** to Windows text files **permit_header.txt**, **permit_attrib.txt**, **permit_owner.txt**, and **permit_trans.txt**. The latter three files are in comma-delimited CSV format. Finally, macro **MakePermitTrans** launches batch file **permit_datafeed.bat**, which appends these four text files into one called **datafeed.txt** that is ready for upload to PacFIN.

RecFIN BDS Datafeed (sub-process 8)

As mentioned previously, there are two BDS datafeeds, one for PacFIN and one for RecFIN. The RecFIN datafeed is pretty simple in that it just assembles many data elements from BDS tables **HEADER**, **FISHDTL**, **ANNULI**, and **VESSEL** into a very wide flat table called **RecFINdatafeed**, then exports it in CSV format and zips it. The datafeed specification includes nearly all of the data fields from the first three tables, but without duplication of key fields. For clarification, the ODBC connection for the BDS **VESSEL** table in the Access database has the alias **VESSEL_BDS** to distinguish it from the CTL **VESSEL** table. It's used to obtain vessel names associated with vessel IDs.

Macro **MakeBDSdfRecFIN** controls the RecFIN BDS datafeed process. **qryMakeBDSRecFIN** uses sub-query **qrySelectRecFINheader** to pull and reformat data from the **HEADER** table, and pulls data directly from FISHDTL and ANNULI, placing the results in table **RecFINdatafeed**. Next, **MakeBDSdfRecFIN** uses the *ImportExportText* macro action to export the contents of **RecFINdatafeed** to Windows file **RecFINdatafeed.csv** in comma-delimited CSV format. Finally, the macro launches batch file **recfin_datafeed.bat**, which simply zips **RecFINdatafeed.csv** using the **7za.exe** utility, resulting in file **recfin.zip**. Unlike the previous datafeeds, there is no automatic upload process for the RecFIN BDS datafeed. WDFW staff must email the file **recfin.zip** to the RecFIN data administrator. However, it's my understanding that there's an automated process to load the data on the RecFIN side and send a receipt email back to WDFW staff.

Buyer Contact Data for NMFS (sub-process 9)

For a few years, WDFW received a request once or twice a year for dealer/buyer contact information so the NMFS buyback program (i.e., Sustainable Fisheries Act fishing capacity reduction) could contact them about revenues owed. It is uncertain if NMFS is still requesting this information or if another source was found (possibly built their own buyer database). Regardless, a brief description of the datafeed process is provided below.

The buyer data is delivered to the NMFS as an Access database containing four tables: **tblBuyerData**, **tblBuyerPhone**, **tblBuyerSpecies**, and **tblSpeciesList**. Table **tblBuyerData** contains the name and address of buyers along with their license ID as a key field. Table **tblBuyerPhone** is just a list of buyer phone numbers with license ID as a key field. Table

**tblBuyerSpecies** is a many-to-many crosswalk between buyer license IDs and WDFW three-digit species codes indicating which species the buyer has purchased. Table **tblSpeciesList** is the WDFW fish ticket species list. All data is taken from **LiFT 2000** using both fish ticket and licensing tables. Unlike all previous datafeeds, this one doesn't use **DatafeedYear**, instead selecting all currently active buyers. There are several queries and temporary tables used in the process, but these are mainly just for cleaning up and formatting the output. The resulting tables are inserted directly in another Access database called **WDFWBuyerData.mdb**, overwriting any existing tables. Finally, the batch file **buyer_datafeed.bat** is run to zip the Access database. As with the RecFIN datafeed, WDFW staff must email the datafeed to the recipient.

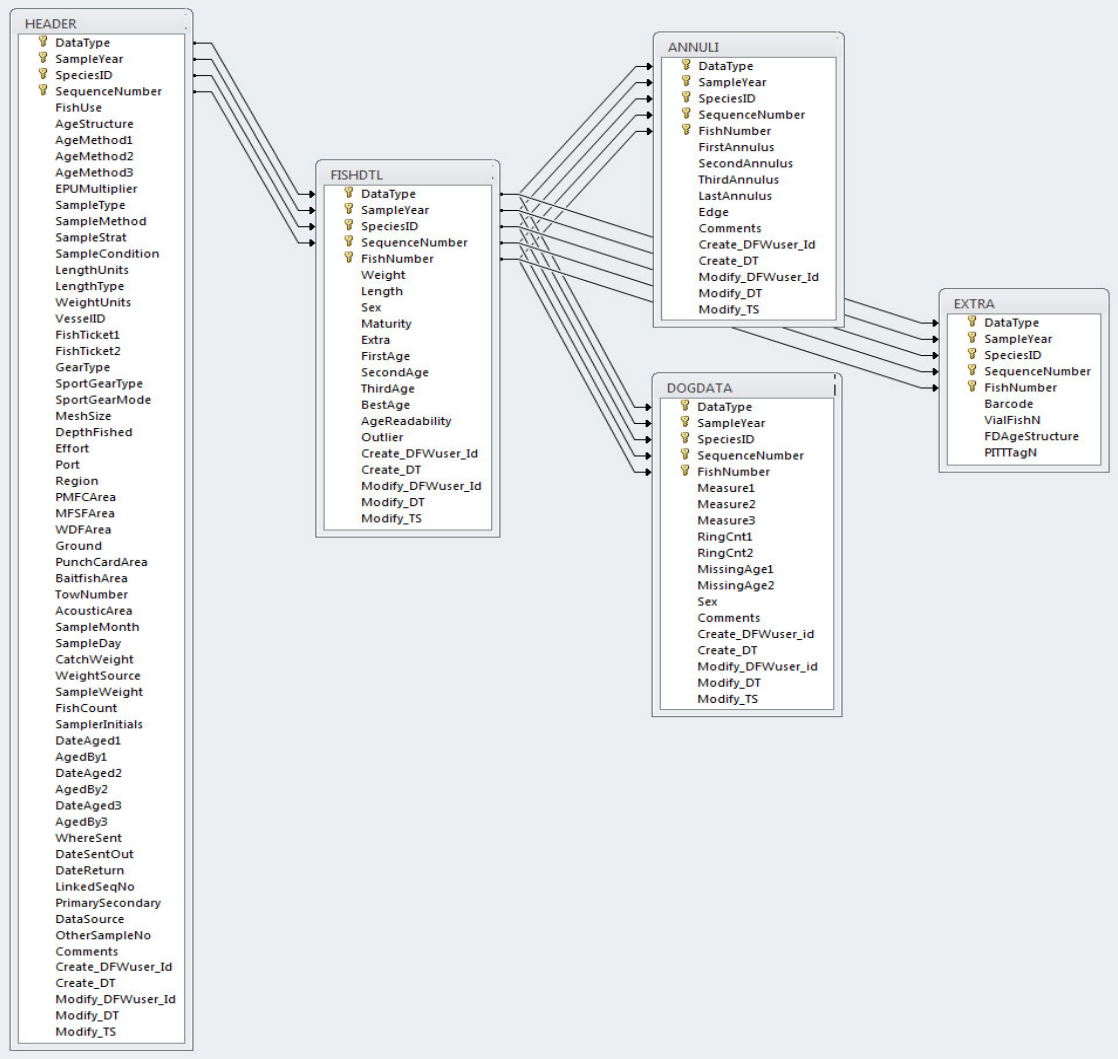
This concludes the text documentation to accompany the datafeed process flow diagram. The current PacFIN datafeed specifications are provided in Appendix K.

# Appendix B: System Documentation for Biological Data System

This document provides system documentation for the Biological Data System (BDS) client/server database in SQL Server and data entry application in MS Access. Other documentation covers the day-to-day operation and administration of the BDS and how it's used in the PacFIN datafeed. For readability, all program code is attached in appendices rather than being embedded in the main body of this document. Windows and SQL Server object names are in italic and Access object names are in boldface so they can be spotted easily.

Data Model

The BDS resides in a SQL Server database called *MFBDS* at server address *Dfwdbolysql2\GISProd* on the agency intranet. The logical relationships of the main BDS tables are shown in the diagram below, key fields labeled with the key icon. These tables use composite, natural keys for uniqueness of rows rather than surrogate keys.



The *HEADER* table contains data describing samples (species, date, location, sample type, etc.). The *FISHDTL* table contains "fish detail" data describing individual fish in the sample (length, weight, age, etc.). The three supplemental tables contain data as described below:

*ANNULI* – annular distance measurements for special fish ageing projects (e.g., lingcod)
*DOGDATA* – some special data elements needed for ageing spiny dogfish
*EXTRA* – some seldom-used data elements relating to individual fish

BDS is the most complex marine resources database for these reasons: 1) it has five main data tables—two primary tables and three supplemental tables—as well as 20 code list look-up tables; 2) there are actually three datasets included in the BDS—commercial, sport, and research—which have many data elements in common, but also some that are unique to the dataset; and 3) there are several versions of the data entry form depending on which BDS dataset and which of the three supplemental tables the user needs to enter data for. The BDS is also the largest marine resources database with more than 44,000 header rows, more than 1.6 million fish detail rows, and with data going back to 1954. Appendix L contains more detailed descriptions of the main data tables, including the data type and size of each field and the particulars of table relationships.

In addition to the main data tables, there are 20 look-tables for code lists used during data entry:

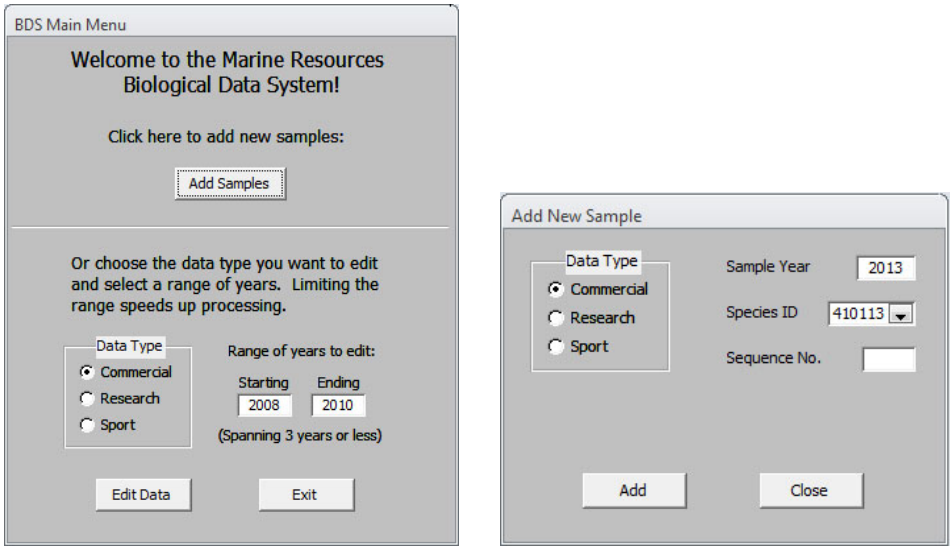
*AGEMETH* – fish ageing methods for *HEADER* fields *AgeMethod1*, *AgeMethod2*, and *AgeMethod3*
*AGESTRUC* – fish ageing structures for *HEADER* field *AgeStructure*
*DATSOURC* – sample data sources for *HEADER* field *DataSource*
*GEAR* – commercial and research gears for *HEADER* field *GearType*
*LENTYPE* – length types for *HEADER* field *LengthType*
*LENUNITS* – length units for *HEADER* field *LengthUnits*
*OUTLIER* – codes for *FISHDTL* field *Outlier*, only used for special projects and not included in forms
*PORTS* – port codes for *HEADER* field *Port*
*READABIL* – age readability codes for *FISHDTL* field *AgeReadability*
*REGION* – region codes for *HEADER* field *Region*
*SAMPMETH* – sample methods for *HEADER* field *SampleMethod*
*SAMPSTRA* – sample stratification types for *HEADER* field *SampleStrat*
*SEX* – sex codes for *FISHDTL* field *Sex*
*SPECIES* – fish species IDs and corresponding names for *HEADER* field *SpeciesID*
*SPGRMODE* – sport gear modes (e.g., private boat, charter) for *HEADER* field *SportGearMode*
*SPGRTYPE* – sport gear types (e.g., hook and line, dip net) for *HEADER* field *SportGearType*
*USE* – fish uses (e.g., human food, bait) for *HEADER* field *FishUse*
*VESSEL* – vessel IDs and corresponding names for *HEADER* field *VesselID*
*WTSOURCE* – weight sources for *HEADER* field *WeightSource*

*WTUNITS* – weight units for *HEADER* field *WeightUnits*

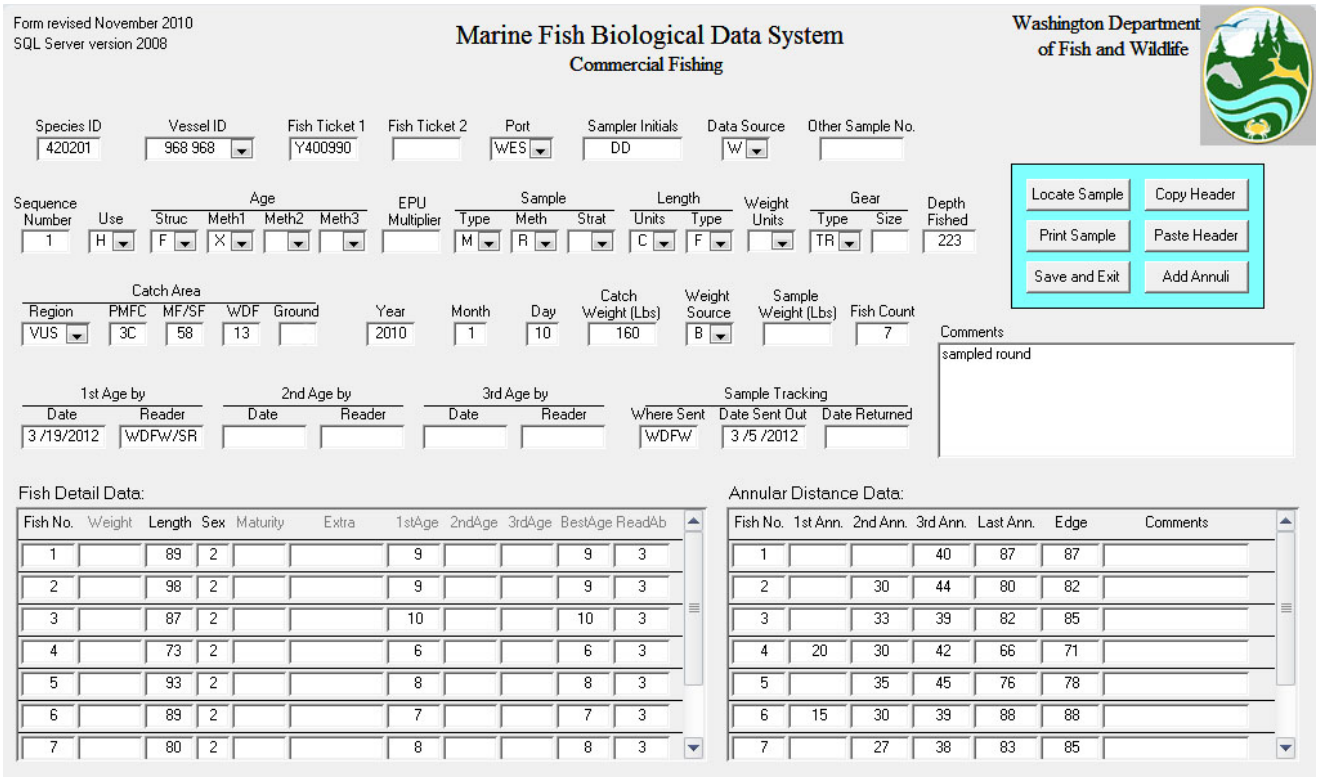
The contents of these look-up tables, including the full list of codes and their descriptions, are included in Appendix M. Only an excerpt from the *VESSEL* table is included because it's quite large with more than 1,000 rows.

Data Entry Form

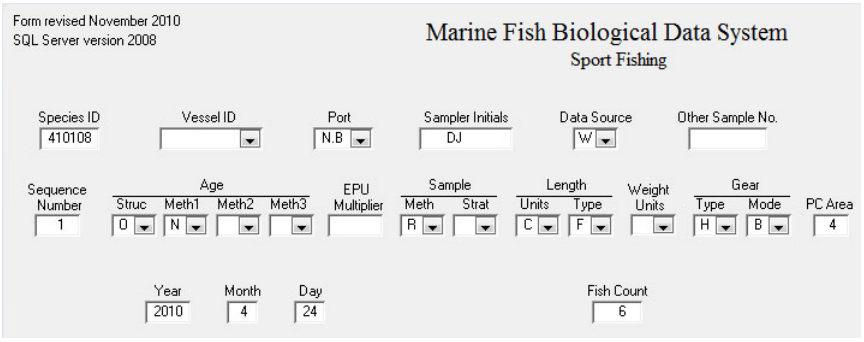
When the Access data entry application is opened, a small start-up form appears. This form asks the user to select a BDS dataset and a range of years for the data entry task. Because the BDS contains so much data, selecting a range of years improves performance of the form during data entry. This is accomplished using the **DoCmd.OpenForm** method with a **WhereCondition** argument, based on the range of years entered. The start-up form, **frmStartUp**, is shown below. This version of **frmStartUp** allows the addition of new samples to the database, a task that's only performed by the sample custodian. The small form for adding new samples, **frmLocSamp**, is also shown below. There's another version of **frmStartUp** that doesn't include the **Add Samples** button. That one is used by all staff other than the sample custodian.

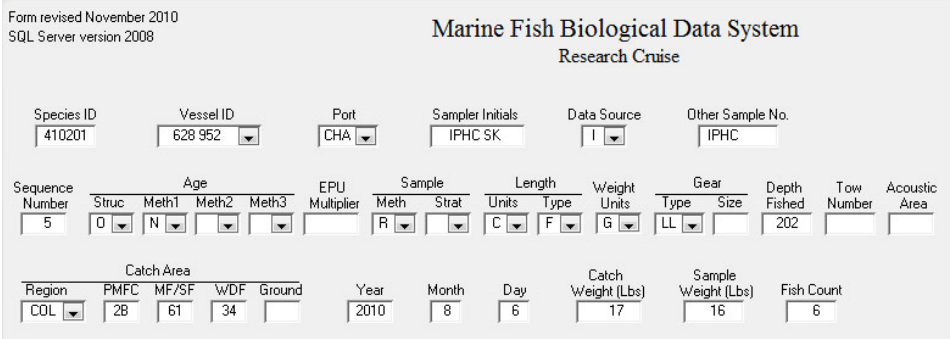



As mentioned previously, there are several versions of the BDS data entry form. The one most commonly used, **frmCommerc**, is shown below.

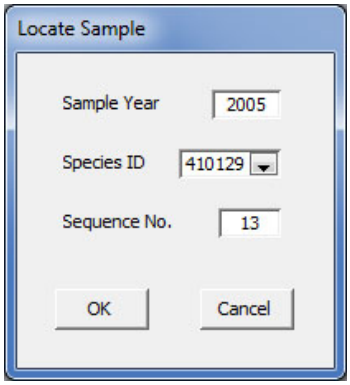



This form has header information (i.e., overall sample descriptive information) customized for commercial BDS samples. There are two other BDS forms with header information customized for sport and research samples, respectively. To conserve space in this document, only the part of these two other forms that's different from the commercial form is shown below.



One feature shared by all the data entry forms is the ability to locate samples easily in the database by pressing the **Locate Sample** button and using the small sample selection form that pops up. Here's what that form, **frmLocSamp**, looks like.




Once selected, the cursor in the *HEADER* table will move to that sample, assuming that it's within the range of years selected in the start-up form. If the sample doesn't exist in the BDS or is outside the range of years selected at start-up, a message will appear indicating the sample wasn't found and the cursor will stay on the current sample.

The other variations of the data entry form pertain to the three supplemental data tables—*ANNULI*, *DOGDATA*, and *EXTRA*—that the data entry task involves. The **Commercial Fishing** form shown previously includes the subform for annular distance data, called **frmAnnuli**. There are two other subforms, used for specific data entry tasks that can appear in the lower right of the data entry form. The subform **frmDogfish** is used to enter special dogfish ageing data. The subform **frmExtra** is used to enter some extra data elements that are not encountered very often. All three subforms contain rows that have a one-to-one relationship with the fish detail rows that appear in the lower left of the data entry form. Besides allowing a wider range of information to be entered for individual fish, the three supplemental data tables allow the BDS to make more efficient use of disk space. If these additional data elements were simply added to the *FISHDTL* table, it would contain a lot of empty cells. The two other subforms that can appear in the lower right of a data entry form are shown below.

Dogfish Data:

| Fish No. | Measure1 | Measure2 | Measure3 | RingCnt1 | RingCnt2 | Comments |
|---|---|---|---|---|---|---|
| 1 | 4.76 | 3.81 | 2.79 | 29 | 26 | |
| 2 | 5.23 | 3.6 | 1.96 | 18 | 11 | |
| 3 | 4.57 | 3.83 | 2.76 | 26 | 17 | |
| 4 | 4.76 | 3.31 | 2.31 | 43 | 37 | |
| 5 | 4.75 | 3.73 | 2.92 | 23 | 18 | |
| 6 | 4.86 | 3.41 | 1.83 | 40 | 28 | |
| 7 | 4.8 | 3.66 | 2.79 | 24 | 18 | |

Extra Data:

| Fish No. | Barcode | VialFishN | FDAgeStruc | PITTTagN |
|---|---|---|---|---|
| 1 | 100309706 | | | |
| 2 | 100309709 | | | |
| 3 | 100309711 | | | |
| 4 | 100309707 | | | |
| 5 | 100309713 | | | |
| 6 | 100309708 | | | |
| 7 | 100309710 | | | |

If either of these subforms is included in the data entry form in lieu of the annular distance subform, the label on the button in the lower right of the cyan-colored button area will say **Add Dogfish Data** or **Add Extra Data**, depending on which subform is included. These subforms start out being unpopulated with rows for a given sample, and this button must be clicked to generate a set of empty rows that correspond to the fish detail rows already in the sample. After empty rows are created, data can be entered into them.

Embedded VBA Code

One major difference between the BDS data entry forms and the Coastal Trawl Logbook System (CTLS) data entry form is that the BDS forms have quite a lot of embedded VBA code, whereas the CTLS form has very little, relying mostly on default Access form functionality. The BDS was developed first as a stand-alone application in Access and was later migrated to a client/server version using SQL Server for data storage. Consequently, actions such as those performed by the CTLS stored procedure *fill_missing_data* were handled differently because Access doesn't have the equivalent of a stored procedure that can be run automatically every night. At the time the BDS Access application was developed, the strategy was to make the data entry form as "intelligent" as possible with time-saving data entry features and data quality control operations, all of which involved a substantial amount of VBA programming. The best way to explain the actions performed by the embedded VBA code is to summarize them in outline form and grouped by the form they're contained in. All of these actions are performed real-time while the form is being used. This list does not include input masks, validation rules, and other built-in features of Access forms. All of this embedded VBA code is included in Appendix N.

**frmCommerc**:

- Allow user to invoke locate sample subform, then move cursor to selected sample
- Allow copy and paste of the header contents
- Check age structure and method codes to make sure they're compatible
- If age structure = 'N', skip over age method fields
- If sample method <> 'S', skip over stratification type field
- Validate area codes entered to see if they're compatible
- Fill marine fish/shellfish area, PMFC area, and region from WDFW area and ground code
- Allow user to print hard copy of sample
- Allow user to create empty rows for supplemental tables *ANNULI*, *DOGDATA*, or *EXTRA* (one for each fish detail row in the current sample)
- Allow saving current sample and exiting the form

**frmFishDtl**:

- Automatically generate next fish number when a new row is added
- Update fish count field in header to match maximum fish number in detail subform
- Allow user to toggle columns on and off for easier data entry (by double-clicking column label)

The VBA code for **frmResearch** is essentially the same as **frmCommerc**. The VBA code in **frmSport** is the same as **frmCommerc**, but lacks the code for filling the commercial catch areas because sport samples have only punch card area. The supplemental data form **frmExtra** has only the code for toggling columns on and off. The small forms **frmStartUp**, **frmLocSamp**, and **frmAddSamp** all contain some VBA code, but not very much and it's specific to the limited tasks they perform.

Change Tracking

The agency standards in place at the time the BDS was migrated to SQL Server require that certain change tracking fields be included and maintained in the main data tables. These fields are: *Create_DFWuser_id*, *Create_DT*, *Modify_DFWuser_id*, *Modify_DT*, and *Modify_TS*. The first two track when a row was added to the table and by whom and the last three track when a row was last modified and by whom. The latter field is a timestamp that's populated automatically by SQL Server; however, the binary data stored in it has never actually been used for anything. For the other four fields, code must be added to data management applications to populate them. After some experimentation, it was determined that the best way to implement this was to use built-in SQL Server events and triggers to capture the essential change tracking data to an intermediate table, and then apply this information to the main data tables using a daily batch process. This proved to be much more efficient than trying to update the change tracking fields real-time, which bogged down the data entry process considerably (the table joins required to do real-time updates were the problem). Besides inserts and updates, deletes are also tracked, but this information cannot be stored in the row being deleted.

Here is how the change tracking process works for the BDS. The insert, update, and delete events execute triggers with names ending in *ITrig*, *UTrig*, and *DTrig*, respectively, stored with the two main data tables, *HEADER* and *FISHDTL*, and two of the supplemental data tables, *ANNULI* and *DOGDATA*. The SQL code in these triggers captures the essential change tracking data to table *AUDIT*. The fields in the *AUDIT* table are: *TableID*, *ActionType*, *DataType*, *SampleYear*, *SpeciesID*, *SequenceNumber*, *FishNumber*, *UserName*, and *ActionDT*. The values stored in *TableID* are 'H', 'F', 'A', or 'D', which identify the target data table as *HEADER*, *FISHDTL*, *ANNULI*, or *DOGDATA*, respectively. The values stored in *ActionType* are 'I', 'U', or 'D', which identify the change action as insert, update, or delete, respectively. The next five fields record the key fields of the new or changed row in the data table. The next two fields store the user ID and date/time associated with the change event. Appendix O contains a more detailed description of the *AUDIT* table, including the data type and size of each field. The SQL code for all the change tracking triggers can be found in Appendix O.

The last part of the change tracking process is to take the data stored in the *AUDIT* table and apply it to the main data tables once each day during non-work hours. This is done by establishing joins to the four data tables using the table ID and the key field values, then applying the change actions using the action type, user ID, and date/time. This is only done for insert and update actions. After the data

tables are successfully updated, these rows are purged from the *AUDIT* table. As mentioned previously, this process doesn't work in the case of deletions because the row in question no longer exists in the data table. So the record of these deletions is just left in the *AUDIT* table. The SQL stored procedure for this batch update process, called *update_change_tracking*, can be found at the end of Appendix O.

There's one issue worth noting regarding change tracking, and that's the problem of preventing the *update_change_tracking* process from itself triggering an update event. If something isn't done about this, the "updates" resulting from the nightly batch process will overwrite the real change tracking data in fields *Modify_DFWuser_id* and *Modify_DT* on the next nightly update. This is handled by a small table called *UTRIG* that has only one row and one integer field called *TrackUpdates* that will have a value of *0* or *1*. When the *update_change_tracking* process runs, *TrackUpdates* is set to *0* temporarily, and then changed back to *1* after the process is completed. In all four *UTrig* triggers, there's an *IF* statement that allows a change tracking event to be recorded only if *TrackUpdates = 1*. This solves the problem of self-generated update events being recorded during execution of *update_change_tracking*.

This concludes the system documentation for the BDS. As mentioned previously, other completed documentation covers the day-to-day operation and administration of the BDS and the role it plays in the PacFIN datafeed process. It should be noted that the BDS is the only comprehensive marine resources database that includes all historical data. The other two datasets, CTLS and rockfish species comps, contain only recent data and should be considered more of a data entry front end to PacFIN than a true database.

# Appendix C: Fish Ticket and Logbook Reconciliation Summary

While logbooks report accurate information for where market category is caught, the pounds caught are recorded as the skipper's estimation known as hailed weight. Fish tickets for this catch provide much more accurate information on pounds caught during the trip, but do not detail where it is caught. The purpose of this program is to relate fish tickets to logbooks and then distribute the fish ticket weight over related logbook tows. This is known as the fish ticket/logbook reconciliation and aims to provide the most accurate weight landed of market category with most accurate spatial distribution.

The "lbk_set_apounds.sql" program is conceptually divided into six steps

---

Step 1: Find implicit WDFW LBK-FT relationships for trips not in lbk_ftid
Step 2: Create WDFW species table optimized for this process
Step 3: Combine explicit and implicit relationships into one table
Step 4: Calculate apounds where species match exists
Step 5: Calculate apounds where species match doesn't exist
Step 6: If unable to calculate apounds, set apounds to hpounds

---

Step 1: Find implicit WDFW LBK-FT relationships for trips not in lbk_ftid

WDFW sends fish ticket IDs along with the related logbook IDs in the logbook transactions (denoted as type 4 in the logbook transaction type). These are considered by PacFIN to be explicit relationships because we explicitly report associated fish tickets to logbook trips in this transaction. We can report these explicit relationships because the CTLS entry application is built to include up to three related fish tickets for the logbook being entered. These explicit relationships appear in the PacFIN table lbk_ftid. Sometimes WDFW cannot report these explicit relationships. For example, if related fish tickets were not entered by person entering logbook data in CTLS, PacFIN will not receive explicitly related fish tickets for these logbooks. Implicit relationships are the fish ticket to logbook relationships that weren't reported by WDFW, but determined with this program by PacFIN. The lbk_ftid table includes both explicit and implicit relationships.

In this step, PacFIN builds implicit relationships for all logbooks that do not appear in the lbk_ftid table as having an explicitly related fish ticket; in other words, logbooks that WDFW didn't relate with fish tickets in the logbook transactions. PacFIN builds these implicit relationships using fish ticket date and vessel with log book tow dates and vessel information. Step one is divided into the following steps:

> 1a. Deletes records from lbk_catch table where apounds were calculated from this program previously.
>
> 1b. Updates remaining records in lbk_catch to null in apounds field in order to reflect the most accurate, current data from this program.

1c. Generates temporary table with logbook ids that are not found in lbk_ftid table.
1d. With logbook ids pulled from step 1c, this program matches corresponding fish ticket data based on matching date and vessel.

Step 2: Create WDFW species table optimized for this process

In this step, PacFIN creates a table selecting only species that exist in lbk_catch for the year (&1) they are running this batch for. This purposefully does not necessarily include all species from related fish ticket information.

Step 3: Combine explicit and implicit relationships into one table

This step combines explicit lbk-ft relationships (those coming from datafeed) with the implicit relationship that were just created in this program. Step 3 is divided into the following steps:

3a. Deletes records in lbk_ftid table that were created from implicit relationships using this program previously.

3b. Inserts new implicit lbk-ft relationships (created in step 1) into lbk_ftid table.

3c. Labels the implicit relationships as such.

At this point, the program has replaced old lbk/ft relationship with new relationships using current data. Now, lbk_ftid contains all logbook trips with associated fish tickets, if able to be determined. All that remains is distributing fish ticket landed weight over related logbooks. In short, the rest of this program calculates proportions of hailed pounds by trip/market category and distributes related fish ticket landed weight of the market category based on those proportions. If the related fish tickets have a market category recorded that does not appear in the related logbook trip, the program distributes the weight over all tows of that trip evenly.

Step 4: Calculate apounds where species match exists

Step 4 is divided into the following steps:

4a. PacFIN summarizes FTL weight by trip_id and spcode where FT-LBK relationship exists in lbk_ftid table. We can only distribute fish ticket landed weight over logbooks if an implicit or explicit relationship exists and market category exists in both fish ticket and logbook.

4b. PacFIN distributes summarized FTL weight (step 4a) for species that exist in lbk_catch table using ratio of hpounds in logbook catch divided by total hpounds within trip to distribute across tows containing each species.

Step 5: Calculate apounds where species match doesn't exist

Step 5 is divided in the following steps:

5a. PacFIN summarizes FTL weight by trip_id and spcode where FT-LBK relationship exists in lbk_ftid table and FT primary key only appears once in lbk_ftid table and FTL species doesn't exist in the set of lbk_catch rows for the corresponding trip_id.

5b. Distribute summarized FTL weight for species that don't exist in lbk_catch table using even distribution across all tows without weighing on effort or hpounds ratio.

Step 6: If unable to calculate apounds, set apounds to hpounds

Apounds is calculated for logbooks only if there is a related fish ticket (explicit or implicit). In cases where a fish ticket cannot be related to a logbook trip, apounds cannot be calculated. This could happen if logbooks and/or fish tickets are missing information that is required to determine relationships such as missing vessel or date data. Also, during the matching of fish tickets to logbooks for implicit relationships, there are cases where duplicate fish ticket primary keys are created. This information is not used, thus apounds is not calculated.

Where no FT-LBK relationship exists in lbk_ftid table, PacFIN sets apounds equal to hpounds and sets apounds_calculated flag to 'N' to indicate inability to calculate apounds from FTL weight.

List of PacFIN tables that are used by this program:

| Table Name | Type of Access |
|---|---|
| ft | SELECT |
| ftl | SELECT |
| lbk_catch | SELECT, INSERT, UPDATE, DELETE |
| lbk_ftid | SELECT, INSERT, UPDATE, DELETE |
| lbk_tow | SELECT |
| lbk_trip | SELECT |
| lbk_sp | SELECT |

Here are brief descriptions of the temporary tables used by this program:

| Table Name | Description |
|---|---|
| lbk_salbs_catch_ft1_YYYY | FTL summary where species matches in lbk_catch |
| lbk_salbs_catch_ft2_YYYY | FTL summary where no species match in lbk_catch |
| lbk_salbs_norel_ftid_YYYY | Similar to lbk_ftid, implicit relationships |
| lbk_salbs_norel_trip_YYYY | Contains all WDFW trips not in lbk_ftid |
| lbk_salbs_sp_YYYY | Similar to lbk_sp, WDFW species only |

The PL/SQL code:

```
PROMPT lbk_set_apounds &1
PROMPT
PROMPT
PROMPT
PROMPT DELETE FROM lbk_catch (source = 'F': rows from FT/FTL)
DELETE FROM lbk_catch ca
WHERE  agid = 'W'
AND    EXISTS (SELECT tr.ryear
           FROM   lbk_trip tr
           WHERE  tr.trip_id = ca.trip_id
           AND    tr.ryear = &1)
AND    source = 'F'

PROMPT UPDATE lbk_catch (source = 'L': rows from LBK)
UPDATE lbk_catch ca
SET    apounds = NULL,
     source = 'L',
     apounds_calculated = NULL
WHERE  agid = 'W'
AND    EXISTS (SELECT tr.ryear
           FROM   lbk_trip tr
           WHERE  tr.trip_id = ca.trip_id
           AND    tr.ryear = &1)

CREATE TABLE lbk_salbs_norel_trip_&1
CREATE TABLE lbk_salbs_norel_trip_&1 (
 TRIP_ID                 NUMBER,
 DRVID                   VARCHAR2(8),
 DDAY                    DATE,
 RDAY                    DATE,
 MINTOWDATE                  DATE,
 MAXTOWDATE                   DATE)
/
PROMPT
PROMPT
PROMPT INSERT INTO lbk_salbs_norel_trip_&1
INSERT INTO lbk_salbs_norel_trip_&1
SELECT tr.trip_id, tr.drvid, tr.dday, tr.rday, t.mintowdate, t.maxtowdate
FROM   lbk_trip tr,
     (SELECT trip_id, MIN(tow_date) AS mintowdate, MAX(tow_date) AS maxtowdate
```

```
      FROM   lbk_tow tw
      WHERE  agid = 'W'
      AND    EXISTS (SELECT tr.ryear
                     FROM   lbk_trip tr
                     WHERE  tr.trip_id = tw.trip_id
                     AND    tr.ryear = &1)
      GROUP BY trip_id) t
WHERE  tr.agid = 'W'
AND    tr.ryear = &1
AND    tr.trip_id = t.trip_id
AND    NOT EXISTS (SELECT lf.trip_id
                   FROM   lbk_ftid lf
                   WHERE  lf.trip_id = tr.trip_id)
/
PROMPT
PROMPT
PROMPT CREATE UNIQUE INDEX lbk_salbs_norel_trip_x1_&1
CREATE UNIQUE INDEX lbk_salbs_norel_trip_x1_&1
ON lbk_salbs_norel_trip_&1 (trip_id)
/
PROMPT
PROMPT
PROMPT CREATE INDEX lbk_salbs_norel_trip_x2_&1
CREATE INDEX lbk_salbs_norel_trip_x2_&1
ON lbk_salbs_norel_trip_&1 (drvid)
/
PROMPT
PROMPT
PROMPT DROP TABLE lbk_salbs_norel_ftid_&1
DROP TABLE lbk_salbs_norel_ftid_&1
/
PROMPT
PROMPT
PROMPT CREATE TABLE lbk_salbs_norel_ftid_&1
CREATE TABLE lbk_salbs_norel_ftid_&1 (
 TRIP_ID         NUMBER,
 TOWNUM          NUMBER,
 AGID            CHAR(1),
 FTID            VARCHAR2(8),
 TICKET_DATE     DATE,
 PARGRP          CHAR(1),
 DRVID           VARCHAR2(8),
 DDAY            DATE,
 RDAY            DATE)
/
PROMPT
```

```
PROMPT
PROMPT INSERT INTO lbk_salbs_norel_ftid_&1
INSERT INTO lbk_salbs_norel_ftid_&1
SELECT t.trip_id, NULL, f.agid, f.ftid, f.tdate, f.pargrp, f.drvid,
     t.dday, t.rday
FROM   lbk_salbs_norel_trip_&1 t, ft f
WHERE  f.agid = 'W'
AND    t.drvid = f.drvid
AND    t.rday IS NOT NULL
AND    t.rday-NVL(t.dday,t.rday) BETWEEN 0 AND 14
AND    f.tdate BETWEEN NVL(t.dday,t.rday) AND t.rday
ORDER BY f.drvid, f.tdate, t.trip_id
/
COMMIT
/
PROMPT
PROMPT
PROMPT
PROMPT DROP TABLE lbk_salbs_sp_&1
DROP TABLE lbk_salbs_sp_&1
/
PROMPT
PROMPT
PROMPT CREATE TABLE lbk_salbs_sp_&1
CREATE TABLE lbk_salbs_sp_&1 AS
SELECT lbk_spcode, category
FROM   lbk_sp
WHERE  agid = 'W'
AND    lbk_spcode IN (SELECT DISTINCT spcode
                      FROM   lbk_catch ca
                      WHERE  agid = 'W'
                      AND    EXISTS (SELECT tr.ryear
                                     FROM   lbk_trip tr
                                     WHERE  tr.trip_id = ca.trip_id
                                     AND    tr.ryear = &1))
/
PROMPT
PROMPT
PROMPT CREATE UNIQUE INDEX lbk_salbs_sp_x1_&1
CREATE UNIQUE INDEX lbk_salbs_sp_x1_&1
ON lbk_salbs_sp_&1 (lbk_spcode)
/
PROMPT
PROMPT
PROMPT CREATE UNIQUE INDEX lbk_salbs_sp_x2_&1
CREATE UNIQUE INDEX lbk_salbs_sp_x2_&1
```

```
ON lbk_salbs_sp_&1 (category)
/
COMMIT
/
PROMPT
PROMPT
PROMPT
PROMPT
PROMPT DELETE FROM lbk_ftid (source = 'F': rows from FT/FTL)
DELETE FROM lbk_ftid l
WHERE  agid = 'W'
AND    EXISTS (SELECT tr.ryear
              FROM   lbk_trip tr
              WHERE  tr.trip_id = l.trip_id
              AND    tr.ryear = &1)
AND    source = 'F'
/
PROMPT
PROMPT
PROMPT UPDATE lbk_ftid (source = 'L': rows from LBK)
UPDATE lbk_ftid l
SET    source = 'L'
WHERE  agid = 'W'
AND    EXISTS (SELECT tr.ryear
              FROM   lbk_trip tr
              WHERE  tr.trip_id = l.trip_id
              AND    tr.ryear = &1)
AND    source IS NULL
/
PROMPT
PROMPT
PROMPT INSERT INTO lbk_ftid
INSERT INTO lbk_ftid
SELECT trip_id, townum, agid, ftid, ticket_date, pargrp, 'F'
FROM   lbk_salbs_norel_ftid_&1
/
COMMIT
/
PROMPT
PROMPT
PROMPT
PROMPT
PROMPT DROP TABLE lbk_salbs_catch_ft1_&1
DROP TABLE lbk_salbs_catch_ft1_&1
/
PROMPT
```

```
PROMPT
PROMPT CREATE TABLE lbk_salbs_catch_ft1_&1

CREATE TABLE lbk_salbs_catch_ft1_&1 AS
SELECT l.trip_id, s.lbk_spcode, s.category,
     SUM(f.landed_wt*NVL(f.factor,1)) AS weight
FROM   ftl f, lbk_ftid l, lbk_salbs_sp_&1 s
WHERE  f.agid = l.agid
AND    f.ftid = l.ftid
AND    f.year = TO_NUMBER(TO_CHAR(l.ticket_date,'YYYY'))
AND    f.month = TO_NUMBER(TO_CHAR(l.ticket_date,'MM'))
AND    f.day = TO_NUMBER(TO_CHAR(l.ticket_date,'DD'))
AND    f.pargrp = l.pargrp
AND    f.category = s.category
AND    f.agid = 'W'
AND    f.landed_wt IS NOT NULL
AND    EXISTS (SELECT f.trip_id
          FROM   lbk_ftid f
          WHERE  f.trip_id = l.trip_id
          AND    EXISTS (SELECT f2.agid,f2.ftid,f2.ticket_date,f2.pargrp
                    FROM   lbk_ftid f2
                    WHERE  f2.agid = f.agid
                    AND    f2.ftid = f.ftid
                    AND    f2.ticket_date = f.ticket_date
                    AND    f2.pargrp = f.pargrp
                    GROUP BY f2.agid,f2.ftid,f2.ticket_date,f2.pargrp
                    HAVING COUNT(*) = 1))
AND    EXISTS (SELECT ca.trip_id
          FROM   lbk_catch ca
          WHERE  ca.trip_id = l.trip_id
          AND    ca.spcode = s.lbk_spcode)
GROUP BY l.trip_id, s.lbk_spcode, s.category
/
PROMPT
PROMPT
PROMPT CREATE UNIQUE INDEX lbk_salbs_catch_ft1_x_&1
CREATE UNIQUE INDEX lbk_salbs_catch_ft1_x_&1
ON lbk_salbs_catch_ft1_&1 (trip_id, lbk_spcode)
/
PROMPT
PROMPT
PROMPT UPDATE lbk_catch SET apounds = distribution based on hpounds ratio
UPDATE lbk_catch ca
SET    apounds =
     round(
     (ca.hpounds/(SELECT SUM(ca2.hpounds)
```

```
            FROM   lbk_catch ca2
            WHERE  ca2.trip_id = ca.trip_id
            AND    ca2.spcode = ca.spcode
            AND    ca2.hpounds IS NOT NULL)) *
      NVL((SELECT f.weight
         FROM   lbk_salbs_catch_ft1_&1 f
         WHERE  f.trip_id = ca.trip_id
         AND    f.lbk_spcode = ca.spcode),0),
      2),
      apounds_calculated = 'Y'
WHERE  agid = 'W'
AND    EXISTS (SELECT tr.ryear
           FROM   lbk_trip tr
           WHERE  tr.trip_id = ca.trip_id
           AND    tr.ryear = &1)
AND    hpounds IS NOT NULL
AND    EXISTS (SELECT s.category
           FROM   lbk_salbs_sp_&1 s
           WHERE  s.lbk_spcode = ca.spcode)
AND    EXISTS (SELECT f.trip_id
           FROM   lbk_salbs_catch_ft1_&1 f
           WHERE  f.trip_id = ca.trip_id)
/
COMMIT
/
PROMPT
PROMPT
PROMPT
PROMPT DROP TABLE lbk_salbs_catch_ft2_&1
DROP TABLE lbk_salbs_catch_ft2_&1
/
PROMPT
PROMPT
PROMPT CREATE TABLE lbk_salbs_catch_ft2_&1
CREATE TABLE lbk_salbs_catch_ft2_&1 AS
SELECT l.trip_id, s.lbk_spcode, s.category,
      SUM(f.landed_wt*NVL(f.factor,1)) AS weight
FROM   ftl f, lbk_ftid l, lbk_salbs_sp_&1 s
WHERE  f.agid = l.agid
AND    f.ftid = l.ftid
AND    f.year = TO_NUMBER(TO_CHAR(l.ticket_date,'YYYY'))
AND    f.month = TO_NUMBER(TO_CHAR(l.ticket_date,'MM'))
AND    f.day = TO_NUMBER(TO_CHAR(l.ticket_date,'DD'))
AND    f.pargrp = l.pargrp
AND    f.category = s.category
AND    f.agid = 'W'
```

```
AND    f.landed_wt IS NOT NULL
AND    EXISTS (SELECT f.trip_id
          FROM   lbk_ftid f
          WHERE  f.trip_id = l.trip_id
          AND    EXISTS (SELECT f2.agid,f2.ftid,f2.ticket_date,f2.pargrp
                    FROM   lbk_ftid f2
                    WHERE  f2.agid = f.agid
                    AND    f2.ftid = f.ftid
                    AND    f2.ticket_date = f.ticket_date
                    AND    f2.pargrp = f.pargrp
                    GROUP BY f2.agid,f2.ftid,f2.ticket_date,f2.pargrp
                    HAVING COUNT(*) = 1))
AND    NOT EXISTS (SELECT ca.trip_id
              FROM   lbk_catch ca
              WHERE  ca.trip_id = l.trip_id
              AND    ca.spcode = s.lbk_spcode)
AND    EXISTS (SELECT tr.ryear
          FROM   lbk_trip tr
          WHERE  tr.trip_id = l.trip_id
          AND    tr.ryear = &1)
GROUP BY l.trip_id, s.lbk_spcode, s.category
/
PROMPT
PROMPT
PROMPT CREATE UNIQUE INDEX lbk_salbs_catch_ft2_x_&1
CREATE UNIQUE INDEX lbk_salbs_catch_ft2_x_&1
ON lbk_salbs_catch_ft2_&1 (trip_id, lbk_spcode)
/
PROMPT
PROMPT
PROMPT INSERT INTO lbk_catch
INSERT INTO lbk_catch
     (trip_id, townum, spcode, disposition, condition, grade, hpounds, apounds,
      apounds_wdfw, warning, agid, source, apounds_calculated)
SELECT tw.trip_id, tw.townum, f.lbk_spcode, 'U', 'U', 'U', NULL,
      round(f.weight/(SELECT COUNT(*) FROM lbk_tow t2 WHERE t2.trip_id = tw.trip_id),2),
      NULL, '000000000', 'W', 'F', 'Y'
FROM   lbk_salbs_catch_ft2_&1 f, lbk_tow tw
WHERE  f.trip_id = tw.trip_id
/
COMMIT
/
PROMPT
PROMPT
PROMPT
PROMPT
```

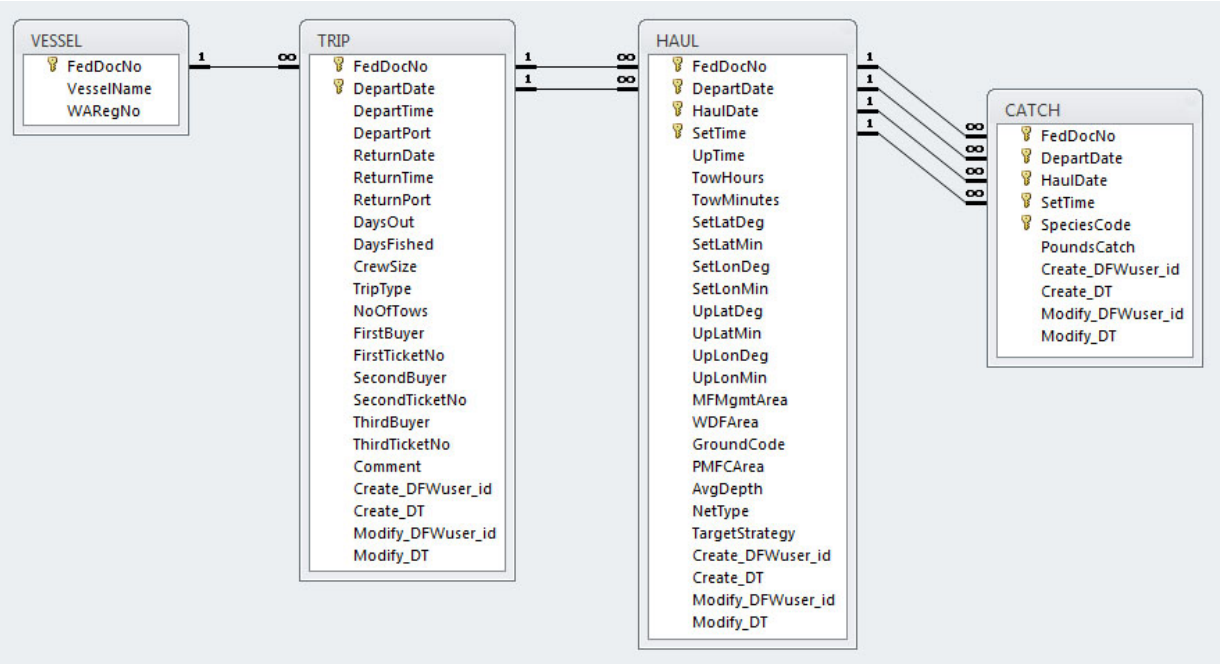
```
PROMPT UPDATE lbk_catch
UPDATE lbk_catch ca
SET    apounds = hpounds,
       apounds_calculated = 'N'
WHERE  agid = 'W'
AND    source = 'L'
AND    apounds_calculated IS NULL
AND    EXISTS (SELECT tr.ryear
               FROM   lbk_trip tr
               WHERE  tr.trip_id = ca.trip_id
               AND    tr.ryear = &1)
/
COMMIT
/
DROP TABLE lbk_salbs_norel_trip_&1;
DROP TABLE lbk_salbs_norel_ftid_&1;
DROP TABLE lbk_salbs_sp_&1;
DROP TABLE lbk_salbs_catch_ft1_&1;
DROP TABLE lbk_salbs_catch_ft2_&1;
!date
EXIT
```

# Appendix D: System Documentation for Coastal Trawl Logbook System

This document provides system documentation for the Coastal Trawl Logbook System (CTLS) client/server database in SQL Server and data entry application in MS Access. Other documentation covers the day-to-day operation and administration of the CTLS and how it's used in the PacFIN datafeed. For readability, program code is attached in appendices rather than being embedded in the main body of this document. Windows and SQL Server object names are in italic and Access object names are in boldface so they can be spotted easily.

Data Model

The CTLS resides in a SQL Server database called *MFCTL* at server address *Dfwdbolysql2\GISProd* on the agency intranet. The logical relationships of the main CTLS tables are shown in the diagram below, key fields labeled with the key icon. These tables use composite, natural keys for uniqueness of rows rather than surrogate keys.




Strictly speaking, the *VESSEL* table is at the top of the hierarchy and table relationships can be described as follows: trips belong to vessels, hauls belong to trips, and catch records belong to hauls. However, for practical purposes and because there's only one row per vessel, the *VESSEL* table can also be viewed as a look-up table for trips. This is how table relationships are presented to the user in the data entry form, i.e., vessels as a look-up table used in a couple of drop-down lists. Appendix P contains more detailed descriptions of these tables, including the data type and size of each field and the particulars of table relationships.

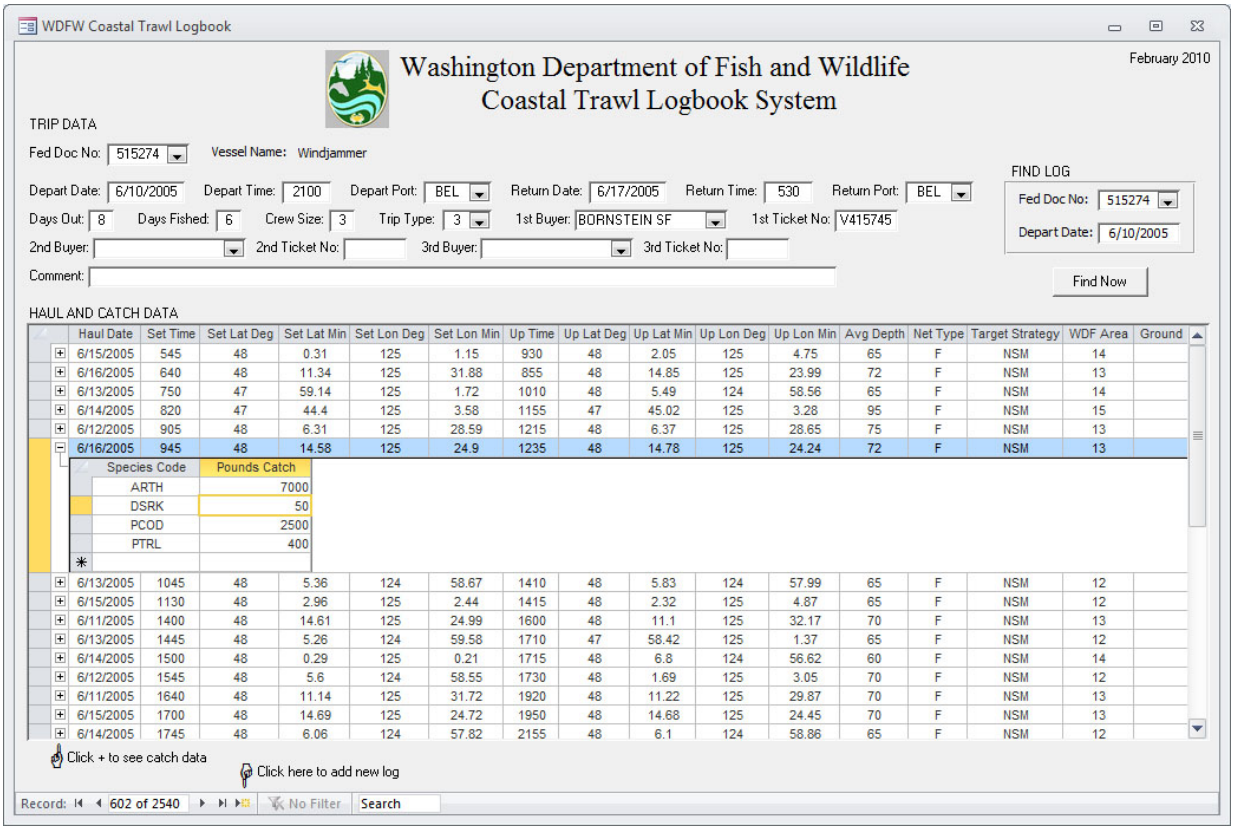
In addition to the main data tables, there are seven look-tables for code lists used during data entry:

*AREA* – the list of catch areas
*GEAR* – the list of trawl gears
*GROUND* – the ground codes for the catch areas
*MARKET* – the list of market categories
*PORT* – the list of WDFW ports
*TARGET* – the list of target strategies
*TRIPTYPE* – the list of trip types

The contents of these look-up tables, including the full list of codes and their descriptions, are included in Appendix Q. An example excerpt from the *VESSEL* table is also included for completeness, due to its ambiguous nature as both a main data table and a look-up table.

Data Entry Form

The CTLS data entry form, **WDFW Coastal Trawl Logbook**, is launched automatically when the Access file *ctls2010.mdb* is opened. The user interface looks like this:



From a developer standpoint, the form is pretty simple and mostly uses the built-in features of Access forms. It contains four embedded sub-forms: **HAUL Subform**, **CATCH Subform**, **VESSEL Subform**, and **FINDLOG Subform**. The relationship of the **HAUL** and **CATCH** sub-forms to each other and the main form matches the data model discussed previously. The **CATCH** sub-form is embedded in the **HAUL** sub-form which is embedded in the main form. The intrinsic table relationships allow the form to function properly, so that catch records appear under their respective hauls and haul records appear with their associated trips. All of the fields in the haul and catch rows that require a code value have drop-down lists—populated by the aforementioned look-up tables—that appear when the cursor lands in that field. Other fields have some validation rules, such as allowable date and numeric ranges..

The features of the data entry form discussed so far all use built-in Access form functionality. One feature worth noting is the column-wise display of haul rows and the drop-down list of catch rows that appear when you click the + sign. The two sub-forms don't actually have a tabular or datasheet view of the data when viewed in design mode. They just have the default columnar view with fields listed one after the other. The trick is to set *Default View = Datasheet* in the sub-form's properties. The sub-form then appears as a datasheet in the main form and you can even adjust the width of columns while running the form. The + sign feature for the catch rows also appears automatically.

The only thing about the form that doesn't use default functionality is the way vessels are handled. The **VESSEL** sub-form is a true embedded sub-form, but it's only used to display the name of the selected vessel in a non-editable field in the upper left of the form. **Fed Doc No** in the upper left is actually a field in the **TRIP** table, which uses the *VESSEL* look-up table as a drop-down list. Selecting a vessel in this part of the form sends the user to the first trip for that vessel chronologically in the database. The problem is there may be many years of data for that vessel in the database, so a better way is needed to locate recent logs that are still being worked on. That's the purpose of the **FINDLOG** sub-form, which is more correctly an embedded dialog box because it has no direct data links to other tables represented in the form. The **FINDLOG** sub-form is linked to a small local table also called **FINDLOG** that has only two fields and one row. The fields are **FedDocNo** and **DepartDate** as displayed in the form. The **FedDocNo** field also uses the *VESSEL* table as a drop-down list. When the user presses the **Find Now** button, some VBA code is used to move the cursor in the *TRIP* table to the selected vessel and depart date. This is the only VBA code embedded in the data entry form and it can be found in Appendix T. Also, when the data entry form is closed and then re-opened, it uses the contents of **FINDLOG** to return to the same log that was last edited.

Change Tracking

The agency standards in place at the time the CTLS was migrated to SQL Server require that certain change tracking fields be included and maintained in the main data tables. These fields are: *Create_DFWuser_id*, *Create_DT*, *Modify_DFWuser_id*, and *Modify_DT*. The first two track when a row was added to the table and by whom, and the last two track when a row was last modified and by whom. Code must be added to data management applications to automatically populate these fields. After some experimentation, it was determined that the best way to implement this was to use built-in SQL Server events and triggers to capture the essential change tracking data to an intermediate table, and then apply this information to the main data tables using a daily batch process. This proved to be much more efficient than trying to update the change tracking fields real-time, which bogged down the data entry process considerably. Besides inserts and updates, deletes are also tracked, but this information cannot be stored in the row being deleted.

Here is how the change tracking process works for the CTLS. The insert, update, and delete events execute triggers with names ending in *ITrig*, *UTrig*, and *DTrig*, respectively, stored with each of the three main data tables. The SQL code in these triggers captures the essential change tracking data to table *AUDIT*. The fields in the *AUDIT* table are: *TableID*, *ActionType*, *FedDocNo*, *DepartDate*, *HaulDate*, *SetTime*, *SpeciesCode*, *UserName*, and *ActionDT*. The values stored in *TableID* are 'T', 'H', or 'C', which identify the target data table as *TRIP*, *HAUL*, or *CATCH*, respectively. The values stored in *ActionType* are 'I', 'U', or 'D', which identify the change action as insert, update, or delete, respectively. The next five fields record the key fields of the new or changed row in the main data table. The next two fields store the user ID and date/time associated with the change event. Appendix P contains a more detailed description of the *AUDIT* table, including the data type and size of each field. The SQL code for all the change tracking triggers can be found in Appendix R.

The last part of the change tracking process is to take the data stored in the *AUDIT* table and apply it to the main data tables once each day during non-work hours. This is done by establishing joins to the three data tables using the table ID and the key field values, then applying the change actions using the action type, user ID, and date/time. This is only done for insert and update actions. After the main data tables are successfully updated, these rows are purged from the *AUDIT* table. As mentioned previously, this process doesn't work in the case of deletions because the row in question no longer exists in the main data table. So the record of these deletions is just left in the *AUDIT* table. The SQL stored procedure for this batch update process, called *update_change_tracking*, can be found at the end of Appendix R.

There's one issue worth noting regarding change tracking, and that's the problem of preventing the *update_change_tracking* process from itself triggering an update event. If something isn't done about this, the "updates" resulting from the nightly batch process will overwrite the real change tracking data in fields *Modify_DFWuser_id* and *Modify_DT* on the next nightly update.

This is handled by a small table called *UTRIG* that has only one row and one integer field called *TrackUpdates* that will have a value of *0* or *1*. When the *update_change_tracking* process runs, *TrackUpdates* is set to *0* temporarily, and then changed back to *1* after the process is completed. In all three *UTrig* triggers, there's an *IF* statement that allows a change tracking event to be recorded only if *TrackUpdates = 1*. This solves the problem of self-generated update events being recorded during execution of *update_change_tracking*.

Fill Missing Data

Another important part of the CTLS operation is the process of filling some derived data fields in the main tables using data entered by users into other fields. These derived data fields, which are not visible in the data entry form, are populated by a batch process—a SQL stored procedure called *fill_missing_data*—run once a day during non-work hours and also just prior to a PacFIN data feed. This *fill_missing_data* process is done for two reasons. First, it saves time during data entry. Second, it ensures that the data in these fields is absolutely correct. For example, it's much easier and safer for the computer to calculate days fished from the dates of the hauls rather than have the user enter this value based on what the fisher recorded in the log. It's quite easy for fishers to make mistakes for calculated values like this and they often do. Here are the actions performed by *fill_missing_data*:

- Fill days fished, number of tows, and tow times
- Fill WDFW area and ground code from lat/lon values
- Fill marine fish management area and PMFC area from WDFW area

During execution of the *fill_missing_data* procedure, tracking of update events needs to be disabled temporarily as explained for the *update_change_tracking* process above. The code for the *fill_missing_data* stored procedure can be found in Appendix S.

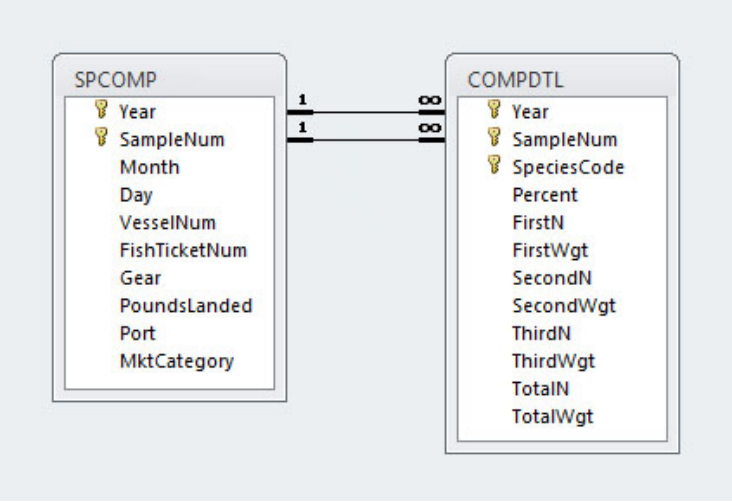
This concludes the system documentation for the CTLS. As mentioned previously, other completed documentation covers the day-to-day operation and administration of the CTLS and the role it plays in the PacFIN datafeed process. It's worth noting that: a) more years of our logbook data reside in PacFIN than in the CTLS, and b) logbook adjusted catch is now computed on the PacFIN side. For these reasons, it's more appropriate to consider the CTLS a data entry front end to PacFIN than a stand-alone database.

# Appendix E: System Documentation for Species Composition Dataset

This document provides system documentation for the rockfish species composition dataset (SCDS) located on SQL Server and its data entry user interface in MS Access. Other documentation covers the day-to-day operation and administration of the SCDS and how it is used in the PacFIN datafeed. For readability, most program code associated with the SCDS is attached in appendices rather than being embedded in the main body of this document. Windows and SQL Server object names are in italic and Access object names are in boldface so they can be spotted easily.

Data Model

The SCDS resides with the Coastal Trawl Logbook System (CTLS) in a SQL Server database called *MFCTL* at server address *Dfwdbolysql2\GISProd* on the agency intranet. This was done for two reasons: 1) it is not a large enough dataset to warrant its own SQL Server database, and 2) it has many things in common with the CTLS. For example, SCDS samples are typically collected at the same time trawl logs are obtained from fishers, and the SCDS and CTLS both describe the trawl fishery (the SCDS also describes the set line fishery). The SCDS has only two main data tables and they have a simple parent/child relationship as shown in the diagram below with key fields labeled using the key icon. These tables use composite, natural keys for uniqueness of rows rather than surrogate keys.




This simple parent/child relationship is reflected in the data entry form. The SCDS allows for entry of up to three subsamples of data for fish counts (N) and weights. This part of the data model is deliberately denormalized with fields for first, second, and third values. There are no change tracking fields in the main data tables of the SCDS because data entry is mainly done by

one person. Appendix F contains more detailed descriptions of these tables, including the data type and size of each field and the particulars of the table relationship.

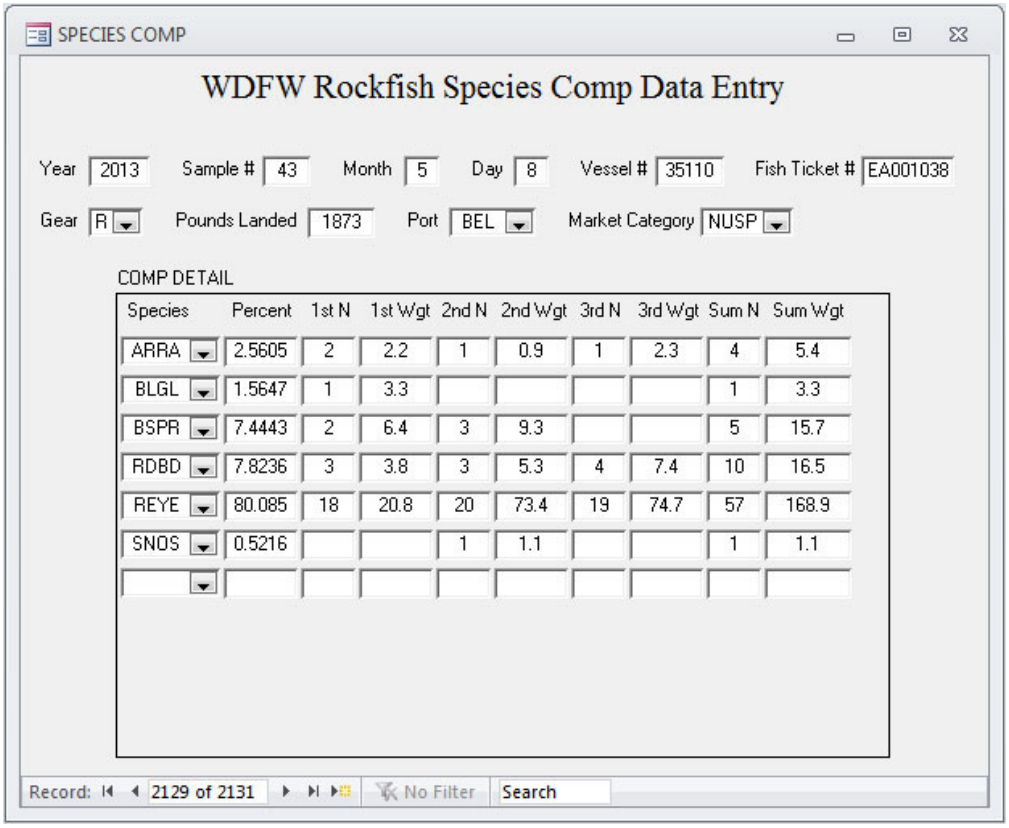
In addition to the main data tables, there are four lookup tables for code lists used during data entry:

*GEAR* – the list of trawl gears, plus set line
*PORT* – the list of WDFW ports, their corresponding PacFIN ports, and port groups
*RFMKTCAT* – the list of rockfish market categories
*ROCKFISH* – the list of rockfish species codes

The contents of these look-up tables, including the full list of codes and their descriptions, are included in Appendix G.

Data Entry Form

The SCDS data entry form, **Species Comp Form**, is launched automatically when the Access file *spcomp.mdb* is opened. The user interface looks like this:




From a developer standpoint, the form is pretty simple and mostly uses the built-in features of Access forms. It contains one embedded sub-form: **Comp Detail Subform**. The relationship

from the sub-form to main form matches the data model discussed previously. The intrinsic table relationships allow the form to function properly, so that comp detail records appear with their associated sample. All of the fields that require a code value have drop-down lists, populated by the aforementioned look-up tables.

The features of the data entry form discussed so far all use built-in Access form functionality. The only thing about the form that does not use default functionality is the process of creating sample numbers, which is done with embedded VBA code. When the cursor exits the **Sample #** field, the code automatically finds the next sample number in sequence for that year and inserts it into the field. Here's the code that does that:

```
Sub GetMaxSamp(intSampYr As Integer)
   Dim db As DAO.Database
   Dim rst As DAO.Recordset
   Dim strSampYr As String
   strSampYr = CStr(intSampYr)
   Set db = CurrentDb()
   Set rst = db.OpenRecordset("SELECT [SampleNum] FROM SPCOMP WHERE [Year]=" &
strSampYr)
   If rst.RecordCount > 0 Then
      rst.MoveLast
      intMaxSampNo = rst![SampleNum]
   Else
      intMaxSampNo = 0
   End If
End Sub

Private Sub SampleNum_Exit(Cancel As Integer)
   If IsNull([SampleNum]) Then
      Call GetMaxSamp([Year])
      [SampleNum] = intMaxSampNo + 1
   End If
End Sub
```

Fill Species Comp Data

Like the CTLS, the SCDS has a batch process that runs once a day during non-work hours to populate some derived data fields based on entered data. These fields are visible in the data entry form, but to save time during data entry and to ensure that the values entered in these fields are

correct, an automated process is used. Here are the actions performed by this stored procedure, called *fill_spcomp_data*:

- Calculate and fill total N and total weight from the subsample values
- Calculate and fill percent of each rockfish species in the sample based on total weights

The code for the *fill_spcomp_data* can be found in Appendix H.

This concludes the system documentation for the SCDS. As mentioned previously, other completed documentation covers the day-to-day operation and administration of the SCDS and the role it plays in the PacFIN datafeed process. There is no counterpart to the SCDS in PacFIN, only species comp transactions derived from it, which represent summary level data.

# Appendix F: Species Compositions Data System (SCDS) Main Table Definitions

SPCOMP Table:

**Columns**

| Name | Type | Size |
|---|---|---|
| Year | Integer | 2 |
| SampleNum | Integer | 2 |
| Month | Integer | 2 |
| Day | Integer | 2 |
| VesselNum | Text | 5 |
| FishTicketNum | Text | 8 |
| Gear | Text | 1 |
| PoundsLanded | Double | 8 |
| Port | Text | 3 |
| MktCategory | Text | 4 |

COMPDTL Table:

**Columns**

| Name | Type | Size |
|---|---|---|
| Year | Integer | 2 |
| SampleNum | Integer | 2 |
| SpeciesCode | Text | 4 |
| SpeciesPct | Double | 8 |
| FirstN | Integer | 2 |
| FirstWgt | Double | 8 |
| SecondN | Integer | 2 |
| SecondWgt | Double | 8 |
| ThirdN | Integer | 2 |
| ThirdWgt | Double | 8 |
| TotalN | Integer | 2 |
| TotalWgt | Double | 8 |

**Relationships**

**SPCOMPCOMPDTL**


| | |
|---|---|
| Attributes: | Enforced, Cascade Updates, Cascade Deletes |
| RelationshipType: | One-To-Many |

# Appendix G: CTLS Code List Look-Up Tables

GEAR:

| Gear | Description | AgencyCode | PacFINCode | GearGroup |
|---|---|---|---|---|
| B | Bottom Trawl (Small Footrope) | 36 | GFS | TWL |
| F | Selective Flatfish | 37 | FTS | TWL |
| L | Set Line | 43 | LGL | HKL |
| M | Midwater Trawl | 34 | MDT | TWL |
| R | Roller Trawl | 35 | RLT | TWL |

PORT:

| MFPortCode | Description | WDFPortCode | MFPortCluster | PacFINCode | SpCompPortGrp |
|---|---|---|---|---|---|
| ANA | Anacortes | 105 | ANA | ANA | BEL |
| BEL | Bellingham | 110 | BEL | BLL | BEL |
| BLA | Blaine | 115 | BLA | BLN | BEL |
| BRE | Bremerton | 120 | PSD | OSP | BEL |
| COU | Coupeville | 125 | PSD | ONP | BEL |
| EVE | Everett | 130 | PSD | EVR | BEL |
| F.H | Friday Harbor | 135 | BEL | FRI | BEL |
| L.C | La Conner | 140 | ANA | LAC | BEL |
| LAC | La Conner | 140 | ANA | LAC | BEL |
| N.B | Neah Bay | 150 | N.B | NEA | N.B |
| OLY | Olympia | 155 | PSD | OLY | BEL |
| P.R | Point Roberts | 156 | BLA | ONP | BEL |
| P.A | Port Angeles | 160 | PEN | PAG | BEL |
| P.T | Port Townsend | 165 | PEN | TNS | BEL |
| POU | Poulsbo | 169 | PSD | OSP | BEL |
| SEA | Seattle | 170 | PSD | SEA | BEL |
| SEQ | Sequim | 175 | PEN | SEQ | BEL |
| SHE | Shelton | 180 | PSD | SHL | BEL |
| BRI | Brinnon | 181 | PSD | OWA | BEL |
| TAC | Tacoma | 190 | PSD | TAC | BEL |
| ABE | Aberdeen | 210 | GRH | GRH | WES |
| C.B | Copalis Beach | 230 | GRH | CPL | WES |
| HOQ | Hoquiam | 240 | GRH | GRH | WES |
| L.P | La Push | 260 | GRH | LAP | N.B |
| TAH | Taholah | 290 | GRH | OWC | WES |
| WES | Westport | 295 | GRH | WPT | WES |
| B.C | Bay Center | 310 | WLB | WLB | WES |
| L.B | Long Beach | 320 | WLB | WLB | WES |
| NAS | Naselle | 340 | WLB | WLB | WES |
| RAY | Raymond | 345 | WLB | WLB | WES |
| S.B | South Bend | 350 | WLB | WLB | WES |
| TOK | Tokeland | 360 | WLB | WLB | WES |
| CHI | Chinook | 409 | COL | LWC | WES |
| ILW | Ilwaco | 421 | COL | LWC | WES |
| KEL | Kelso | 423 | COL | OCR | WES |
| VAN | Vancouver | 438 | COL | OCR | WES |
| ALA | Alaska (all ports) | 501 | ALA | AAL | OTH |
| CAL | California (all | 502 | CAL | ACA | OTH |
| ORE | Oregon (all ports) | 503 | ORE | AOR | OTH |

RFMKTCAT:

| MktCategory | Name | PacFINCode |
|---|---|---|
| CNRY | Nom. Canary Rockfish | CNR1 |
| DBRK | Nom. Darkblotched Rockfish | DBR1 |
| LSPN | Nom. Longspine Thornyhead | LSP1 |
| NUSF | Shelf Rockfish | NUSF |
| NUSP | Slope Rockfish | NUSP |
| NUSR | Near-shore Rockfish | NUSR |
| POP | Nom. Pacific Ocean Perch | UPOP |
| SSPN | Nom. Shortspine Thornyhead | SSP1 |
| WDOW | Nom. Widow Rockfish | WDW1 |
| YEYE | Nom. Yelloweye Rockfish | YEY1 |
| YTRK | Nom. Yellowtail Rockfish | YTR1 |

ROCKFISH:

| PacFINCode | SpeciesName |
|---|---|
| ARRA | Aurora Rockfish |
| BCAC | Bacacio |
| BLCK | Black Rockfish |
| BLGL | Blackgill Rockfish |
| BLUR | Blue Rockfish |
| BRWN | Brown Rockfish |
| BSPR | Blackspotted Rockfish |
| CHNA | China Rockfish |
| CLPR | Chilipepper |
| CNRY | Canary Rockfish |
| COPP | Copper Rockfish |
| DBRK | Darkblotched Rockfish |
| GSPT | Greenspotted Rockfish |
| GSRK | Greenstriped Rockfish |
| LSPN | Longspine Thornyhead |
| PGMY | Pygmy Rockfish |
| POP | Pacific Ocean Perch |
| PUGT | Puget Sound Rockfish |
| QLBK | Quillback Rockfish |
| RDBD | Redbanded Rockfish |
| REDS | Redstripe Rockfish |
| REYE | Rougheye Rockfish |
| RSTN | Rosethorn Rockfish |
| SBLY | Shortbelly Rockfish |
| SHRP | Sharpchin Rockfish |
| SLGR | Silvergrey Rockfish |
| SNOS | Splitnose Rockfish |
| SRKR | Shortraker Rockfish |
| SSPN | Shortspine Thornyhead |
| STRK | Stripetail Rockfish |
| TIGR | Tiger Rockfish |
| VRML | Vermilion Rockfish |
| WDOW | Widow Rockfish |
| YEYE | Yelloweye Rockfish |
| YMTH | Yellowmouth Rockfish |
| YTRK | Yellowtail Rockfish |

# Appendix H: Fill Species Comp Data Stored Procedure

```
CREATE PROCEDURE fill_spcomp_data AS

/* * USE DATA ENTERED TO POPULATE OTHER FIELDS */

BEGIN

/* * FILL TOTAL N AND TOTAL WEIGHT */

UPDATE COMPDTL SET TotalN = ISNULL(FirstN,0) + ISNULL(SecondN,0) +
ISNULL(ThirdN,0)
FROM COMPDTL
UPDATE COMPDTL SET TotalWgt = ISNULL(FirstWgt,0) + ISNULL(SecondWgt,0) +
ISNULL(ThirdWgt,0)
FROM COMPDTL

/* * FILL SPECIES PERCENT OF WEIGHT */

CREATE TABLE #spcompsum_tmp
(
Year smallint,
SampleNum smallint,
SampWgtSum float,
)

INSERT INTO #spcompsum_tmp (Year, SampleNum, SampWgtSum)
SELECT Year, SampleNum, SUM(TotalWgt)
FROM COMPDTL
GROUP BY Year, SampleNum

UPDATE COMPDTL SET SpeciesPct = TotalWgt / SampWgtSum * 100
FROM COMPDTL, #spcompsum_tmp
WHERE COMPDTL.Year = #spcompsum_tmp.Year AND COMPDTL.SampleNum =
#spcompsum_tmp.SampleNum

END

GRANT EXECUTE ON fill_spcomp_data TO dbo
```

# Appendix I: SQL Stored Procedures Used for Fish Ticket/Vessel/Buyer Datafeed to PacFIN

**Procedure: spa_pacfin_fishticket_feed.sql**

```
--Authors: G. Konkel and J. Fiat (last modified by P. Weyland and G. Konkel)
--Date: 5/23/2013
--Installation: SQL Server (instead of Sybase)
--Remarks:
--This stored procedure is used to extract fish ticket data from the WDFW
--LIFT2000 database and prepare a data feed for the PacFIN system.
--
--Process description:
--1. Create a cursor from the FISH_TICKET and FISH_TICKET_DETAIL tables
--2. Process each row selected and prepare each field according to the
--        PacFIN system specification
--3. Output table: pacfin_datafeed
--4. Error report: pacfin_errorlog
--
--Once stored with the LIFT2000 database, the procedure can be run by
--simply executing it with a 4-digit year argument.
--
--Note: This script was modified in 11/2004 to accommodate the catch
--category enhancements to the PacFIN specification. Modified again in
--04/2007 and 07/2007 to improve assignment of vessel_used = T/F.
--Modified again in 08/2009 to fix the problem of price = zero being
--converted to null (zero is a valid price) and to add fisher license ID
--to the data feed per the PacFIN system specification of 07/23/2009.
--Modified again in 06/2010 to fix the following issues:
--  Added species 007, aquaculture salmon, to the excluded data types
--  Added a default value of 'U' for product_form and assignment
--  Added round_lbs_price_amt to the 'order by' clause to fix AQUA
--    row consolidation
--  Added a join to the new pacfin_permit table to more easily merge
--    federal permit numbers with FTL rows (instead of a post process)
--  Tweaked the criteria for determining vessel_used = T/F
--Modified again in 05/2013 for SQL Server compatibility:
--  System variable @@sqlstatus replaced with @@fetch_status
--  Added "begin transaction" to implicit transaction with "commit"
--  Revised null-related syntax replacing "= null" with "is null" and
```

```
--    "<> null" with "is not null"
--  Added SET CONCAT_NULL_YIELDS_NULL OFF to fix problem with string
--    concatenation using variables that might be null
------------------------------------------------------------------------

use lift2000
go

if exists (select 1 from sysobjects
     where name = 'spa_pacfin_fishticket_feed')
begin
     drop procedure spa_pacfin_fishticket_feed
end
go

--Stored procedure starts here.
create procedure spa_pacfin_fishticket_feed (@batch_year smallint) as
begin

--Create temporary tables.
create table #pacfin_ft_tmp (pacfin_trans char(45))
create table #pacfin_ftl_tmp (pacfin_trans char(75))

--Suppress messages.
set nocount on

--This controls how nulls are evaluated when concatenating strings.
--When SET CONCAT_NULL_YIELDS_NULL is ON, concatenating a null value with
--a string yields a NULL result. When SET CONCAT_NULL_YIELDS_NULL is OFF,
--concatenating a null value with a string yields the string itself
--(the null value is treated as an empty string).
set concat_null_yields_null off

--Establish cursor for fish ticket table. Filter out some unwanted
--data types. Records types excluded are:
--  Source codes IMPT and INVO
--  Aquaculture finfish
--  Shellfish seed (species 519 and 529)
--  Commercial freshwater
declare ftdtl_curs cursor for
```

```
select
    fish_ticket.fish_ticket_id,
    batch_year,
    batch_num,
    datepart(month, fish_ticket.landing_date) as landing_month,
    datepart(day, fish_ticket.landing_date) as landing_day,
    ticket_data_source_code,
    fish_ticket_num,
    dealer_license_id,
    af_license_id,
    case
        when (port_code is null)
            then '000'
        else convert(char(3), port_code)
    end as port_code,
    case
        when ticket_data_source_code = 'AQUA'
            then 'A'
        when fisher_type_code = 2
            then 'I'
        else 'C'
    end as par_group,
    case
        when fisher_type_code = 2
            then '  '
        else substring(str(days_fished_cnt, 3, 0), 2, 2)
    end as days_fished,
    case
        when (gear_code = '  ' or gear_code is null)
            then '00'
        else gear_code
    end as gear_code,
    case
        when gear_code in ('05','51','52','53','60','62','65','71','76','81','86','94')
        or ticket_data_source_code = 'UNKN'
        or fish_ticket_detail.species_code in ('504','505','508')
            then 'F'
        else 'T'
    end as vessel_used,
    vessel_id,
```

```
license_id,
tribe_num,
tribe_member_num,
catch_area_code,
fish_ticket_detail.species_code,
case
    when catch_desc_code in (1, 7, 13, 16)
        then 'L'
    when catch_desc_code in (2, 3, 4, 5, 8, 9, 10, 11, 14, 17)
        then 'M'
    when catch_desc_code in (6, 12, 15, 18)
        then 'S'
    when catch_desc_code in (19, 28, 29, 30, 31)
        then 'O'
    else 'U'
end as grade,
case
    when (fish_ticket_detail.species_code in ('519','529')
    or fish_ticket_detail.rptd_lbs_type_code = 6)
        then 'O'
    when fish_ticket_detail.species_code between '521' and '525'
        then 'S'
    when fish_ticket_detail.species_code between '860' and '877'
        then 'E'
    when (fish_ticket_detail.species_code = '291'
    and fish_ticket_detail.rptd_lbs_type_code = 2)
        then 'W'
    when fish_ticket_detail.rptd_lbs_type_code = 1
        then 'R'
    when fish_ticket_detail.rptd_lbs_type_code = 2
        then 'G'
    when fish_ticket_detail.rptd_lbs_type_code = 3
        then 'F'
    when fish_ticket_detail.rptd_lbs_type_code = 4
        then 'D'
    when fish_ticket_detail.rptd_lbs_type_code = 5
        then 'H'
    else 'U'
end as condition,
case
```

```
        when ticket_data_source_code = 'EFSZ'
            then 'W'
        when ticket_data_source_code in ('FFRA','TEST')
            then 'I'
        when ticket_data_source_code = 'SEIZ'
            then 'S'
        when ticket_data_source_code in ('C&SF','TKHM')
            then 'P'
        when fish_ticket_detail.species_code in ('122','123','124','125','141','142','577')
            then 'B'
        when fish_ticket_detail.species_code in ('519','529')
            then 'O'
        when catch_desc_code = 20
            then 'D'
        when catch_desc_code = 21
            then 'A'
        when (catch_desc_code between 0 and 19 or catch_desc_code between 28 and 31)
            then 'H'
        else 'U'
    end as disposition,
    round_wgt_conv_fctr_qty,
    fish_ticket_detail.rptd_cnt_type_code,
    rptd_cnt,
    rptd_lbs_qty,
    rptd_price_amt,
    round_lbs_qty,
    round_lbs_price_amt,
    gallon_cnt,
    case
        when ticket_data_source_code in
('AQUA','CAND','COMM','EGGS','ORGN','QUIN','SEIZ')
            then 'C'
        when ticket_data_source_code in ('EFPT','EFSZ')
            then 'E'
        when ticket_data_source_code in ('C&SF','TKHM')
            then 'P'
        when ticket_data_source_code in ('FFRA','TEST')
            then 'R'
        else 'O'
    end as removal_type,
```

```
    case
        when ticket_data_source_code = 'SEIZ'
            then 'F'
        else 'T'
    end as legal_removal,
    case
        when fish_ticket_detail.species_code in ('122','123','124','125','141','142','577')
            then 'B'
        when fish_ticket_detail.species_code in ('519','529')
            then 'O'
        when catch_desc_code = 20
            then 'D'
        when catch_desc_code = 21
            then 'A'
        when (catch_desc_code between 0 and 19 or catch_desc_code between 28 and 31)
            then 'H'
        else 'U'
    end as product_use,
    case
        when ticket_data_source_code in ('EFSZ','SEIZ')
            then 'T'
        else 'F'
    end as overage,
    permit_num
from
    fish_ticket
        inner join fish_ticket_detail
            on fish_ticket.fish_ticket_id = fish_ticket_detail.fish_ticket_id
        left join species_qty_conv_lut
            on fish_ticket_detail.rptd_cnt_type_code = species_qty_conv_lut.rptd_cnt_type_code
            and fish_ticket_detail.species_code = species_qty_conv_lut.species_code
            and fish_ticket_detail.unit_type_code = species_qty_conv_lut.unit_type_code
        left join pacfin_permit
            on fish_ticket.fish_ticket_num = pacfin_permit.ticket_num
where ticket_data_source_code not in ('IMPT','INVO')
    and fish_ticket_detail.species_code not in ('007','200','270','272','519','529','581')
    and fish_ticket_detail.species_code not between '041' and '099'
    and batch_year = @batch_year
order by
    fish_ticket_num,
```

```
    par_group,
    landing_month,
    landing_day,
    port_code,
    gear_code,
    catch_area_code,
    fish_ticket_detail.species_code,
    grade,
    condition,
    disposition,
    rptd_price_amt,
    round_lbs_price_amt

--Fish ticket table image variables and derived variables, which receive
--values for each row read with fetch next. Some PacFIN fields are derived
--during the query for use in sorting the incoming table extract according
--to PacFIN key field values.
declare @fish_ticket_id             integer
declare @batch_num                  smallint
declare @landing_month              tinyint
declare @landing_day                tinyint
declare @ticket_data_source_code    char(4)
declare @fish_ticket_num            char(8)
declare @dealer_license_id          integer
declare @af_license_id              integer
declare @output_license_id          char(6)
declare @port_code                  char(3)
declare @ls_port_code               char(3)
declare @par_group                  char(1)
declare @days_fished                char(2)
declare @gear_code                  char(3)
declare @vessel_used                char(1)
declare @vessel_id                  integer
declare @ls_vessel_id               char(6)
declare @license_id                 integer
declare @fisher_license_id          char(6)
declare @tribe_num                  char(2)
declare @tribe_member_num           integer
declare @ls_tribe_member_num        char(6)
declare @catch_area_code            char(4)
```

```
declare @species_code                char(3)
declare @grade                       char(1)
declare @condition                   char(1)
declare @disposition                 char(1)
declare @round_wgt_conv_fctr_qty     numeric(14,5)
declare @rptd_cnt_type_code          integer
declare @rptd_cnt                    numeric(14,0)
declare @rptd_lbs_qty                numeric(14,0)
declare @rptd_price_amt              numeric(14,5)
declare @round_lbs_qty               numeric(14,0)
declare @round_lbs_price_amt         numeric(14,5)
declare @gallon_cnt                  numeric(14,0)
declare @removal_type                char(1)
declare @legal_removal               char(1)
declare @product_use                 char(1)
declare @overage                     char(1)
declare @permit_num                  char(6)

--Working variables.
declare @err_flag                    tinyint
declare @last_tkt_num                char(7)
declare @last_port_code              char(3)
declare @last_landg_month_num        tinyint
declare @last_landg_day_num          tinyint
declare @err_str                     char(100)
declare @date_str                    char(6)
declare @vessel_type_id_str          char(9)
declare @ft_trans                    char(45)
declare @catch_area                  char(4)
declare @weight_of_catch             char(9)
declare @convert_factor              char(5)
declare @number_of_fish              char(6)
declare @price_per_pound             char(7)
declare @new_weight_of_catch         numeric(14,0)
declare @last_weight_of_catch        numeric(14,0)
declare @new_convert_factor          char(5)
declare @last_convert_factor         char(5)
declare @new_number_of_fish          numeric(14,0)
declare @last_number_of_fish         numeric(14,0)
declare @new_ftl_key_p1              char(22)
```

```
declare @last_ftl_key_p1        char(22)
declare @new_ftl_key_p2         varchar(30)
declare @last_ftl_key_p2        varchar(30)
declare @new_ftl_key            varchar(52)
declare @last_ftl_key           varchar(52)
declare @ftl_trans              varchar(75)
declare @report_trans           char(72)

--Purge output tables.
begin transaction
delete from pacfin_datafeed
delete from pacfin_errorlog
commit

--Open cursor to fish ticket table. Fetch first row. Check for invalid year.
open ftdtl_curs
fetch ftdtl_curs into @fish_ticket_id, @batch_year, @batch_num, @landing_month,
@landing_day, @ticket_data_source_code, @fish_ticket_num, @dealer_license_id,
@af_license_id, @port_code, @par_group, @days_fished, @gear_code, @vessel_used,
@vessel_id, @license_id, @tribe_num, @tribe_member_num, @catch_area_code,
@species_code,
@grade, @condition, @disposition, @round_wgt_conv_fctr_qty, @rptd_cnt_type_code,
@rptd_cnt, @rptd_lbs_qty, @rptd_price_amt, @round_lbs_qty, @round_lbs_price_amt,
@gallon_cnt, @removal_type, @legal_removal, @product_use, @overage, @permit_num
if @@fetch_status = -1
    begin
    print 'No fish ticket data for that year.'
    close ftdtl_curs
    return
    end

--Initialize variables.
select @last_tkt_num = '       '
select @last_port_code = '   '
select @last_weight_of_catch = 0
select @last_number_of_fish = 0
select @last_ftl_key = space(52)

--Begin loop that steps through the fish ticket query result one row at a
--time until the cursor reaches the end of the query.
```

```
while @@fetch_status = 0
begin

--Check for null record, no weight or count. If so, write error message
--and skip to next row.
if ((@rptd_lbs_qty = 0 or @rptd_lbs_qty is null)
    and (@rptd_cnt = 0 or @rptd_cnt is null)
    and (@round_lbs_qty = 0 or @round_lbs_qty is null)
    and (@gallon_cnt = 0 or @gallon_cnt is null))
    begin
    select @err_str = 'Null record, no weight or count: ticket=' +
    @fish_ticket_num + ' year=' + right(str(@batch_year),4) + ' batch=' +
    right(str(@batch_num),3) + ' record=' + right(str(@fish_ticket_id),4)
    insert into pacfin_errorlog (error_msg) values (@err_str)
--Get next record.
    fetch ftdtl_curs into @fish_ticket_id, @batch_year, @batch_num, @landing_month,
    @landing_day, @ticket_data_source_code, @fish_ticket_num, @dealer_license_id,
    @af_license_id, @port_code, @par_group, @days_fished, @gear_code, @vessel_used,
    @vessel_id, @license_id, @tribe_num, @tribe_member_num, @catch_area_code,
@species_code,
    @grade, @condition, @disposition, @round_wgt_conv_fctr_qty, @rptd_cnt_type_code,
    @rptd_cnt, @rptd_lbs_qty, @rptd_price_amt, @round_lbs_qty, @round_lbs_price_amt,
    @gallon_cnt, @removal_type, @legal_removal, @product_use, @overage, @permit_num
    continue
    end

--Attempt to fix the aquaculture problem of multiple port codes per ticket
--by decrementing the day of the month.
if @ticket_data_source_code = 'AQUA'
    begin
    select @err_flag = 0
    if (@fish_ticket_num = @last_tkt_num and @Landing_month = @last_landg_month_num)
        begin
        if @port_code <> @last_port_code
            begin
            if @last_landg_day_num > 1
                select @last_landg_day_num = @last_landg_day_num - 1
            else
                select @err_flag = 1
            select @landing_day = @last_landg_day_num
```

```
            select @last_port_code = @port_code
            end
        else
            select @landing_day = @last_landg_day_num
        end
    else
        begin
        select @last_tkt_num = @fish_ticket_num
        select @last_landg_month_num = @landing_month
        select @last_landg_day_num = @landing_day
        select @last_port_code = @port_code
        end
    if @err_flag = 1
        begin
        select @err_str = 'Aquaculture ticket has too many ports: ticket=' +
        @fish_ticket_num + ' year=' + right(str(@batch_year),4) + ' batch=' +
        right(str(@batch_num),3) + ' record=' + right(str(@fish_ticket_id),4)
        insert into pacfin_errorlog (error_msg) values (@err_str)
        end
    end

--Convert date for FT transaction to PacFIN format.
select @date_str = right(str(@batch_year),2) + right(str(@landing_month+100),2) +
right(str(@landing_day+100),2)

--Convert tribe member number.
select @tribe_num = isnull(@tribe_num, '  ')
select @tribe_member_num = isnull(@tribe_member_num, 0)
select @ls_tribe_member_num = right(str(@tribe_member_num + 100000),5)

--Clean up and prepare vessel data.
if @vessel_used = 'F' and @species_code <> '508' and @vessel_id is not null
    select @vessel_used = 'T'
if @vessel_id is null or @vessel_id = 0 or @vessel_used = 'F'
    select @ls_vessel_id = '      '
else
    if (@vessel_id < 100000)
        select @ls_vessel_id = right(str(@vessel_id + 100000),5) + ' '
    else
        select @ls_vessel_id = convert(char(6),@vessel_id)
```

```
select @vessel_type_id_str =
    case
        when @vessel_used = 'F'
            then '        '
        when @par_group = 'I'
            then '4' + @tribe_num + @ls_tribe_member_num
        when @ls_vessel_id <> '      '
            then '3' + @ls_vessel_id + '  '
        when @ticket_data_source_code = 'CAND'
            then '5       '
        else 'U       '
    end

--Prepare correct dealer license ID according to ticket data source.
if @ticket_data_source_code = 'AQUA'
    select @output_license_id =
        case
            when @af_license_id < 10000
                then right(str(@af_license_id + 10000),4)
            else
                convert(char(6),@af_license_id)
        end
else
    select @output_license_id =
        case
            when @dealer_license_id < 10000
                then right(str(@dealer_license_id + 10000),4)
            else
                convert(char(6),@dealer_license_id)
        end

--Prepare fisher license ID.
select @fisher_license_id =
    case
        when @license_id = 0 or @license_id is null
            then '      '
        when @license_id < 100000
            then right(str(@license_id + 100000),5)
        else
            convert(char(6),@license_id)
```

```
    end

--Convert numerics to strings.
select @ls_port_code = isnull(@port_code,'   ')
select @output_license_id = isnull(@output_license_id,'       ')
select @days_fished = isnull(@days_fished,'  ')

--Build FT transaction and insert into buffer table.
select @ft_trans = 'F&' + @fish_ticket_num + @date_str + @ls_port_code +
    @vessel_used + @vessel_type_id_str + @output_license_id + ' ' +
    @par_group + @days_fished + @fisher_license_id
insert into #pacfin_ft_tmp (pacfin_trans) values (@ft_trans)

--Convert data needed for FTL transaction type to PacFIN format.
select @catch_area =
    case
        when substring(@catch_area_code,1,2) = '  '
            then substring(@catch_area_code,3,1) + '   '
        when substring(@catch_area_code,1,1) = ' '
            then substring(@catch_area_code,2,2) + '  '
        when substring(@catch_area_code,1,3) = 'UNK'
            then 'UNKN'
        else @catch_area_code + ' '
    end

--Select appropriate landing weight, count and price values.
if (@species_code between '521' and '525')
    begin
    select @new_weight_of_catch = isnull(@round_lbs_qty,0)
    select @new_convert_factor = ' 1000'
    select @new_number_of_fish = isnull(@gallon_cnt,0)
    if @round_lbs_price_amt is null
        select @price_per_pound = '       '
    else
        select @price_per_pound = str(@round_lbs_price_amt*1000,7,0)
    end
else
    begin
    select @new_weight_of_catch =
        case
```

```
            when ((@rptd_lbs_qty = 0 or @rptd_lbs_qty is null)
            and @round_lbs_qty > 0 and @round_wgt_conv_fctr_qty > 1)
                then round(@round_lbs_qty/@round_wgt_conv_fctr_qty,0)
            when ((@rptd_lbs_qty = 0 or @rptd_lbs_qty is null)
            and @round_lbs_qty > 0)
                then @round_lbs_qty
            else isnull(@rptd_lbs_qty,0)
        end
    select @new_convert_factor = str(@round_wgt_conv_fctr_qty*1000,5,0)
    select @new_number_of_fish = isnull(@rptd_cnt,0)
    if @rptd_price_amt is null
        select @price_per_pound = '       '
    else
        select @price_per_pound = str(@rptd_price_amt*1000,7,0)
    end

--Prepare fields for output and create record keys.
select @new_ftl_key_p1 = @fish_ticket_num + @species_code + ' ' + @grade +
@condition + @disposition + '2' + @catch_area + @gear_code
select @new_ftl_key_p2 = @price_per_pound + @par_group + @date_str +
'    ' + @removal_type + @legal_removal + @product_use + 'U' + @overage +
'U' + @permit_num
select @new_ftl_key = @new_ftl_key_p1 + @new_ftl_key_p2

--Consolidate rows with same key values.
if (@new_ftl_key = @last_ftl_key or @last_ftl_key = space(52))
    begin
    select @last_weight_of_catch = @last_weight_of_catch + @new_weight_of_catch
    select @last_convert_factor = @new_convert_factor
    select @last_number_of_fish = @last_number_of_fish + @new_number_of_fish
    select @last_ftl_key_p1 = @new_ftl_key_p1
    select @last_ftl_key_p2 = @new_ftl_key_p2
    select @last_ftl_key = @new_ftl_key
    end
else
    begin
    select @weight_of_catch = str(@last_weight_of_catch,9,0)
    select @convert_factor = isnull(@last_convert_factor,' 1000')
    if @last_number_of_fish is null or @last_number_of_fish = 0
        select @number_of_fish = '      '
```

```
    else
        select @number_of_fish = str(@last_number_of_fish,6,0)

--Since this is the last row for these key field values, build FTL transaction
--and insert into buffer table.
    select @ftl_trans = 'L&' + @last_ftl_key_p1 + ' ' + @weight_of_catch +
        @convert_factor + @number_of_fish + @last_ftl_key_p2
    insert into #pacfin_ftl_tmp (pacfin_trans) values (@ftl_trans)
    select @last_weight_of_catch = @new_weight_of_catch
    select @last_convert_factor = @new_convert_factor
    select @last_number_of_fish = @new_number_of_fish
    select @last_ftl_key_p1 = @new_ftl_key_p1
    select @last_ftl_key_p2 = @new_ftl_key_p2
    select @last_ftl_key = @new_ftl_key
    end

--Fetch next row and continue loop.
fetch ftdtl_curs into @fish_ticket_id, @batch_year, @batch_num, @landing_month,
@landing_day, @ticket_data_source_code, @fish_ticket_num, @dealer_license_id,
@af_license_id, @port_code, @par_group, @days_fished, @gear_code, @vessel_used,
@vessel_id, @license_id, @tribe_num, @tribe_member_num, @catch_area_code,
@species_code,
@grade, @condition, @disposition, @round_wgt_conv_fctr_qty, @rptd_cnt_type_code,
@rptd_cnt, @rptd_lbs_qty, @rptd_price_amt, @round_lbs_qty, @round_lbs_price_amt,
@gallon_cnt, @removal_type, @legal_removal, @product_use, @overage, @permit_num

--End cursor loop.
end

--Main loop has ended. With data left from last fetch, build final FTL
--transaction and insert into buffer table. Close cursor.
select @weight_of_catch = str(@last_weight_of_catch,9,0)
select @convert_factor = isnull(@last_convert_factor,'     ')
if @last_number_of_fish is null or @last_number_of_fish = 0
    select @number_of_fish = '      '
else
    select @number_of_fish = str(@last_number_of_fish,6,0)
select @ftl_trans = 'L&' + @last_ftl_key_p1  + ' ' + @weight_of_catch +
@convert_factor + @number_of_fish + @last_ftl_key_p2
```

```
insert into #pacfin_ftl_tmp (pacfin_trans) values (@ftl_trans)

close ftdtl_curs

--Transfer unique FT transactions from buffer table to data feed
--output table.
insert into pacfin_datafeed (pacfin_trans)
select distinct pacfin_trans from #pacfin_ft_tmp

--Transfer FTL transactions from buffer table to data feed output
--table.
insert into pacfin_datafeed (pacfin_trans)
select pacfin_trans from #pacfin_ftl_tmp

--Turn on messages.
set nocount off

return 0

end
go

grant execute on spa_pacfin_fishticket_feed to public
go
```

**Procedure: spa_pacfin_vessel_feed.sql**

```
------------------------------------------------------------------------
--Procedure: spa_pacfin_vessel_feed
--Authors: G. Konkel and J. Fiat (last revision by G. Konkel)
--Date: 6/14/2007
--Installation: Sybase
--Remarks:
--This stored procedure is used to extract vessel data from the WDFW
--LIFT2000 database and prepare a data feed for the PacFIN system.
--
--Process description:
--1. Select vessel and owner data into two temporary tables from VESSEL,
--    VESSEL_LICENSE, PARTICIPANT, PARTICIPANT_LICENSE, and ADDRESS tables
--2. Combine results with outer join (in case owner data is missing)
--    and create cursor for sequential processing
--3. Process each row selected and prepare each field according to the
--    PacFIN specification
--4. Output table: pacfin_datafeed
--
--Once stored with the LIFT2000 database, the procedure can be run by
--simply executing it with a 4-digit year argument.
------------------------------------------------------------------------

use lift2000
go

if exists (select 1 from sysobjects
    where name = 'spa_pacfin_vessel_feed')
begin
    drop procedure spa_pacfin_vessel_feed
end
go

--
create procedure spa_pacfin_vessel_feed (@batch_year smallint) as
begin

--Create temporary tables.
create table #pacfin_vessel_tmp (pacfin_trans varchar(170))
```

```
--Establish cursor for boat reg table. Select only vessels that have
--fish ticket landings for @batch_year.

set nocount on

--Get all vessels for the selected year that match fish ticket vessels.
select distinct vessel.vessel_id,
        vessel.vessel_name,
        vessel.smb_doc_num,
        vessel.length_meas,
        vessel.net_tonnage_meas,
        vessel.hp_meas
      into #pacfin_qry1_tmp
      from vessel inner join vessel_license on vessel.vessel_id = vessel_license.vessel_id
      where exists (select 1 from fish_ticket, fish_ticket_detail
        where batch_year = @batch_year
        and fish_ticket.vessel_id = vessel.vessel_id
        and fish_ticket.fish_ticket_id = fish_ticket_detail.fish_ticket_id
        and ticket_data_source_code <> 'IMPT'
        and ticket_data_source_code <> 'INVO'
        and fish_ticket_detail.species_code not between '041' and '099'
        and fish_ticket_detail.species_code not in ('519','529','200','270','272','581'))
      and datepart(year,vessel_license.active_dt)<=@batch_year
      and isnull(datepart(year,vessel_license.inactive_dt),@batch_year)>=@batch_year

--Get owner data for these vessels.
declare @end_of_year datetime
select @end_of_year = convert(datetime,'Dec 31 '+right(str(@batch_year),4))
select distinct participant_vessel.vessel_id,
       case
            when (participant.participant_last_name is null)
                then participant.business_name
            else participant.participant_last_name + ', ' + participant.participant_first_name
       end as owner_name,
       address.addr1,
       address.city_name,
       address.stprov_code,
       address.postal_code
       into #pacfin_qry2_tmp
```

```
    from participant_vessel inner join participant on participant_vessel.participant_id
        = participant.participant_id inner join address on participant.participant_id
        = address.participant_id
    where exists (select 1 from fish_ticket, fish_ticket_detail
        where batch_year = @batch_year
        and fish_ticket.vessel_id = participant_vessel.vessel_id
        and fish_ticket.fish_ticket_id = fish_ticket_detail.fish_ticket_id
        and ticket_data_source_code <> 'IMPT'
        and ticket_data_source_code <> 'INVO'
        and fish_ticket_detail.species_code not between '041' and '099'
        and fish_ticket_detail.species_code not in ('519','529','200','270','272','581'))
        and datepart(year,participant_vessel.active_dt)<=@batch_year
        and isnull(participant_vessel.inactive_dt,@end_of_year)>= @end_of_year
        and participant_vessel.participation_role_code=4
        and address.addr_type_code=1

--Combine vessel and owner results and create cursor for sequential processing.
declare boat_curs cursor for
select distinct #pacfin_qry1_tmp.*,
    #pacfin_qry2_tmp.owner_name,
    #pacfin_qry2_tmp.addr1,
    #pacfin_qry2_tmp.city_name,
    #pacfin_qry2_tmp.stprov_code,
    #pacfin_qry2_tmp.postal_code
    from #pacfin_qry1_tmp, #pacfin_qry2_tmp
    where #pacfin_qry1_tmp.vessel_id *= #pacfin_qry2_tmp.vessel_id
    order by #pacfin_qry1_tmp.vessel_id

--Boat registration table image variables, which receive values for
--each row read with fetch. Variable boat_reg_num also occurs in the
--fish ticket table.
declare @vessel_id          integer
declare @vessel_name        varchar(20)
declare @smb_doc_num        varchar(10)
declare @length             smallint
declare @net_tonnage        smallint
declare @hp                 smallint
declare @owner              char(32)
declare @Street             char(35)
declare @city               char(17)
```

```
declare @state              char(2)
declare @zip                varchar(10)

--Working variables.
declare @out_trans          varchar(170)
declare @alpha              char(26)
declare @digit              char(10)
declare @hasdigit           tinyint
declare @i                  tinyint

--Output variables.
declare @pf_vessel_id_type      char(1)
declare @last_pf_vessel_id      char(8)
declare @pf_vessel_id           char(8)
declare @pf_plate_number        char(6)
declare @pf_vessel_name         char(37)

--Set constants and initial values.
select @alpha = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
select @digit = '0123456789'
select @last_pf_vessel_id = 'GOOFBALL'

--Open cursor to boat reg table. Fetch first row.
open boat_curs
fetch boat_curs into @vessel_id, @vessel_name, @smb_doc_num,
@length, @net_tonnage, @hp, @owner, @street, @city, @state, @zip

--Begin loop that steps through query result from boat reg table.
--Convert data to PacFIN format. Clean up output as much as possible.
while @@sqlstatus = 0

begin

if (@vessel_id < 100000)
    select @pf_plate_number = right(str(@vessel_id + 100000),5) + ' '
else
    select @pf_plate_number = convert(char(6),@vessel_id)
select @smb_doc_num = isnull(@smb_doc_num,'         ')
select @pf_vessel_name = upper(@vessel_name)
if charindex(substring(@smb_doc_num,1,1),@digit) > 0
```

```
    begin
    if substring(@smb_doc_num,1,2) in ('13','15','17','18','19') or
        substring(@smb_doc_num,1,1) in ('3','8')
        select @pf_vessel_id_type = '5'
    else
        select @pf_vessel_id_type = '1'
    if substring(@smb_doc_num,4,1) = ' '
        select @pf_vessel_id =
        substring(@smb_doc_num,1,3) + substring(@smb_doc_num,5,3) + '    '
    else
        select @pf_vessel_id = @smb_doc_num + '      '
    end
else
    begin
    if charindex(substring(@smb_doc_num,1,1),@alpha) > 0 and
        charindex(substring(@smb_doc_num,3,1),@digit) > 0
        begin
        if substring(@smb_doc_num,1,2) = 'RC'
            begin
            select @pf_vessel_id_type = 'W'
            select @pf_vessel_id = 'WRN' + @pf_plate_number
            end
        else
            begin
            select @pf_vessel_id_type = '2'
            select @pf_vessel_id = @smb_doc_num + '     '
            end
        end
    else
        begin
        if (substring(@vessel_name,1,2) = 'WN' or
            substring(@vessel_name,1,2) = 'OR') and
            charindex(substring(@vessel_name,3,1),@digit) > 0
            begin
            select @pf_vessel_id_type = '2'
            select @pf_vessel_id = @vessel_name + '     '
            if @smb_doc_num = '          ' or
                substring(@smb_doc_num,1,4) = 'NONE'
                select @pf_vessel_name = ''
            else
```

```
            select @pf_vessel_name = rtrim(@smb_doc_num)
        end
    else
        begin
        select @smb_doc_num = @smb_doc_num + '          '
        select @hasdigit = 0
        select @i = 1
        while @hasdigit = 0 and @i <= 10
            begin
            if charindex(substring(@smb_doc_num,@i,1),@digit) > 0
                select @hasdigit = 1
            select @i = @i + 1
            end
        if @hasdigit = 0
            begin
            if substring(@smb_doc_num,1,3) <> substring(@vessel_name,1,3)
                and @smb_doc_num <> '          ' and substring(@smb_doc_num,1,4) <>
'NONE'
                begin
                if substring(@smb_doc_num,1,6) = 'SALMON'
                    select @pf_vessel_id = 'SALMON    '
                if substring(@smb_doc_num,1,6) = 'DRIFTB'
                    or substring(@smb_doc_num,1,7) = 'DRIFT B'
                    select @pf_vessel_id = 'DRIFT BOAT'
                end
            if char_length(@vessel_name) = 0
                select @pf_vessel_name = rtrim(@smb_doc_num)
            else
                begin
                if rtrim(@smb_doc_num) <> rtrim(@Vessel_name)
                    select @pf_vessel_name = rtrim(@vessel_name) +
                        ' (' + rtrim(@smb_doc_num) + ')'
                else
                    select @pf_vessel_name = rtrim(@vessel_name)
                end
            if substring(@smb_doc_num,1,8) = 'CANADIAN'
                begin
                select @pf_vessel_id_type = '5'
                select @pf_vessel_id = 'UNKNOWN'
                end
```

```
                else
                    begin
                    select @pf_vessel_id_type = 'W'
                    select @pf_vessel_id = 'WRN' + @pf_plate_number
                    end
                end
            else
                begin
                select @pf_vessel_id_type = 'U'
                select @pf_vessel_id = @smb_doc_num + '       '
                end
            end
        end
    end

--Build SV transaction and insert into data feed output table.
select @out_trans = 'V&' + @pf_vessel_id_type + @pf_vessel_id +
    @pf_plate_number + '  1' + isnull(right(str(@length),3),'   ') +
    '1 ' + isnull(right(str(@net_tonnage),4),'    ') +
    isnull(right(str(@hp),4),'    ') + right(str(@batch_year),2) +
    @pf_vessel_name + @owner + @street + @city + @state + @zip
if @pf_vessel_id <> @last_pf_vessel_id
      insert into #pacfin_vessel_tmp (pacfin_trans) values (@out_trans)
select @last_pf_vessel_id = @pf_vessel_id

--Fetch next row and continue loop.
fetch boat_curs into @vessel_id, @vessel_name, @smb_doc_num,
@length, @net_tonnage, @hp, @owner, @street, @city, @state, @zip

--End cursor loop.
end

close boat_curs

--Transfer unique vessel transactions from buffer table to data feed output table.
insert into pacfin_datafeed (pacfin_trans)
select distinct pacfin_trans from #pacfin_vessel_tmp

set nocount off
```

```
return 0

--End create procedure spa_pacfin_vessel_feed
end
go

grant execute on spa_pacfin_vessel_feed to public
go
```

**Procedure: spa_pacfin_buyer_feed.sql**

```
--Procedure: spa_pacfin_buyer_feed
--Authors: J. Fiat and G. Konkel (last revision by G. Konkel)
--Date: 4/16/2010
--Installation: Sybase
--Remarks:
--This stored procedure is used to extract license data from the WDFW
--LIFT2000 database and prepare a buyer data feed for the PacFIN system.
--
--Process description:
--1. Create a query from the LICENSE, PARTICIPANT, ANNUAL_LICENSE,
--    and PARTICIPANT_LICENSE tables
--2. Prepare each field according to the PacFIN specification
--3. Output table: pacfin_datafeed
--
--Once stored with the LIFT2000 database, the procedure can be run by
--simply executing it with a 4-digit year argument.
------------------------------------------------------------------------

use lift2000
go

if exists (select 1 from sysobjects
    where name = 'spa_pacfin_buyer_feed')
begin
    drop procedure spa_pacfin_buyer_feed
end
go

--
```

```
create procedure spa_pacfin_buyer_feed (@batch_year smallint) as
begin

declare @out_trans varchar(72)

create table #pacfin_buyer_tmp
(buyer_code varchar(6) null,
 description varchar(58) null)

--Query license tables and convert data to PacFIN specification.
insert into #pacfin_buyer_tmp
select buyer_code =
    case
        when license.license_id < 10000
        then right(str(license.license_id + 10000),4)
        else convert(char(6),license.license_id)
    end,
    ltrim(coalesce(business_name, convert(varchar(60), participant_first_name + ' '
            + participant_mid_name + ' ' + participant_last_name)))
    from annual_license,
        license,
        participant,
        participant_license
    where license.license_id = annual_license.license_id and
        participant_license.license_id = license.license_id and
        participant_license.participant_id = participant.participant_id
            and license.license_type_code in (82, 83, 84, 85, 87)
            and participant_license.participation_role_code = 1
            and annual_license.license_year = @batch_year
            and datepart(year,participant_license.active_dt) <= @batch_year
            and (participant_license.inactive_dt is null or
            datepart(year,participant_license.inactive_dt) >= @batch_year)

insert into pacfin_datafeed
select distinct 'Y&B' + buyer_code +
    case
        when char_length(buyer_code) =  4 then '       '
        else '     '
    end + description as pacfin_trans
from #pacfin_buyer_tmp
```

```
    order by pacfin_trans

return 0

end
go

grant execute on spa_pacfin_buyer_feed to public
go
```

# Appendix J: Area and Species Comp Formulas for PacFIN Datafeed

The proportion of each rockfish species in a stratum is determined from a simple ratio estimator following Cochran (1977), a stratum being defined as a calendar quarter, gear type, port group, and market category. For a given stratum,

let $\omega_{iko}$ be the total weight of rockfish species *o* in sample *k* for market category *i*,
$\omega_{ik}$ be the total weight of sample *k* for market category *i*,
$\Phi_{io}$ be the estimated proportion of rockfish species *o* in market category *i*.

Then the estimated proportion of rockfish species *o* in market category *i* is calculated as

$$\Phi_{io} = \sum_{k} \omega_{iko} \Big/ \sum_{k} \omega_{ik}$$

---

As with species comps, area comps use a simple ratio estimator to assign adjusted catch to catch areas within a stratum, a stratum being defined as a month, trip type (Puget Sound or coastal), port, and market category. For a given stratum,

let $\theta_{ita}$ be the total adjusted catch of market category *i* in logbook trip *t* and catch area *a*,
$\theta_{it}$ be the total adjusted catch of market category *i* in logbook trip *t*,
$\Psi_{ia}$ be the estimated proportion of market category *i* from catch area *a*.

Then the estimated proportion of market category *i* from catch area *a* is calculated as

$$\Psi_{ia} = \sum_{t} \theta_{ita} \Big/ \sum_{t} \theta_{it}$$

---

Cochran, W. G. 1977. Sampling techniques, 3rd edition. John Wiley & Sons, New York, 428 p.

# Appendix K: Current PacFIN Datafeed Specifications

SCOPE OF THIS SYSTEM SPECIFICATION
==========

1. Retain all features of the current (as of November 1990) PacFIN Central Processing System as implemented on the NOAA Fisheries computer system in Seattle.

2. Enhance the current system to include:
   a. Un-reduced fish ticket data for WDFW, ODFW, CDFG for all commercially harvested marine and anadromous species including steelhead, sturgeon, and shad.
   b. Vessel data provided by WDFW, ODFW, CDFG
   c. Coast Guard vessel data provided by NMFS
   d. WDFW, ODFW, CDFG composition data (species, area, average-wt., salmon deliveries, salmon days-fished)

3. The scope specifically excludes: recreational catch; catch of commercial freshwater species, marine discards, and prohibited species; salmon aquaculture harvest; and joint-venture tow records.

SOURCE INPUT SPECIFICATION
==========

All PacFIN data will be input using the following transactions (or record types):

1. AGENCY-CODE-LIST Transaction (TRANS-ID = 'Y')
2. FISH-TICKET Transaction (TRANS-ID = 'F')
3. FISH-TICKET-LINE Transaction (TRANS-ID = 'L')
4. STATE-VESSEL Transaction (TRANS-ID = 'V')
5. CG-VESSEL Transaction (TRANS-ID = 'X') ** NOT IN USE **

A COMPOSITION Transaction can be one of the following:

6. AREA-COMP Transaction (TRANS-ID = 'A')
7. SPECIES-COMP Transaction (TRANS-ID = 'S')
8. AVERAGE-WEIGHT Transaction (TRANS-ID = 'W')
9. EFFORT-COMP Transaction (TRANS-ID = 'E')

10. AGGREGATED-CATCH Transaction (TRANS-ID = 'C')
11. AGGREGATED-EFFORT Transaction (TRANS-ID = 'D')
12. W-O-C PERMIT-ATTRIBUTE Transaction (TRANS-ID = 'Q')
13. W-O-C PERMIT Transaction (TRANS-ID = 'P')
14. W-O-C PERMIT-OWNER Transaction (TRANS-ID = 'R')
15. WOC-OBSERVER Transaction (TRANS-ID = 'B')

The AGGREGATED-CATCH transaction will not be used by WDFW, ODFW, and CDFG. The AGGREGATED-EFFORT transaction will be used by WDFW, ODFW, and CDFG only to input trawl hours.

All data items are either alpha, integer, real, or text. Alpha items are letters and/or numbers left justified with blank fill. Only uppercase letters are allowed. Integer items will be right justified with either zero or blank fill. Real items are right justified and may include a decimal point. Items that may be NULL (i.e. not required) will contain blank characters. Text data items may contain any printable character and the <blank> character.

All data items that specify a 2-digit year use the following 2-digit to 4-digit rule: ... 98 = 1998; 99 = 1999; 00 = 2000; 01 = 2001; ...

All data items are required unless specified otherwise.

Definitions for this Specification Document:

SPECIES-CATEGORY is defined to mean a collection of species. There are two types of species categories:

1. a collection of known species in known proportions (e.g. ORCK => other rockfish)
2. a collection of unknown species in unknown proportions (e.g. URCK => unspecified rockfish)

SPECIES is defined to mean a scientifically defined species from the PacFIN Species List.

TICKET-CATEGORY is defined to mean a species or a collection of species specified on a FISH-TICKET-LINE transaction (i.e. a reporting agency's species code).

MARKET-CATEGORY is defined to be the combination of TICKET-CATEGORY, GRADE,

CONDITION, and DISPOSITON.

TRANS-OP occurs in each transaction type except the aggregated-catch (AGC) and aggregated-effort (AGE) transaction types. The AGC and AGE transaction types use TRANS-TYPE instead. For all transactions the action that will be taken by the central processing software for each TRANS-OP or TRANS-TYPE value is:

if TRANS-OP = ADD (+) or TRANS-TYPE = ADD (+)
then if the corresponding record already exists in the database
    then the input transaction is rejected
    otherwise the corresponding record is added to the database.

if TRANS-OP = CHANGE (*) or TRANS-TYPE = CHANGE (*)
then if the corresponding record exists in the database
    then all data items within the corresponding record that require
        changes are changed
    otherwise the input transaction is rejected.

if TRANS-OP = DELETE (-) or TRANS-TYPE = DELETE (-)
then if the corresponding record exists in the database
    then the corresponding record is deleted
    otherwise the input transaction is rejected.

In addition to the above operators (+,*,-), the change-or-add (&) operator is also valid for TRANS-TYPE in the AGC and AGE transaction types.

If TRANS-TYPE = change-or-add (&)
then if the corresponding record exists in the database
then the transaction is treated as a change (*) transaction
otherwise the transaction is treated as an add (+) transaction.

A PacFIN datafeed will consist of a data file which will include a single report record (must be the first record in the file) and one or more transaction records. This data file need not be sorted in any particular order, except that the report record must be the first record. Any sorting that may be necessary will be accomplished by the central processing system.

The report record consists of the following data items:

| DATA ITEM | COLS. | RANGE OF VALUES; DESCRIPTION |
|---|---|---|
| SOURCE | 1-5 | agency acronym;<br>values: ADFG, AKR, CDFG, DFO, NWAFC, ODFW, WDFW, AFSC |
| | 6 | *NOT USED* |
| RPT-DATE | 7-12 | YYMMDD (e.g. 901012); this is the date of report; (i.e. the date the datafeed was generated); two datafeeds from the same data source may not have the same date. |
| CONT-FLAG | 13-13 | = "C" => this datafeed is a continuation report |
| COMM/DESC | 14-72 | 59 char comment or description regarding this periodic report; optional |

The AGENCY-CODE-LIST transaction consists of the following data items or attributes:

| DATA ITEM | COLS | DESCRIPTION; RANGE OF VALUES |
|---|---|---|
| TRANS-ID | 1 | set to "Y" => CODE-LIST transaction |
| TRANS-OP | 2 | "+" => add, "*" => change, "-" => delete |
| CODE-TYPE | 3 | alpha; "S" => agency species code<br>"A" => agency area code<br>"G" => agency gear code<br>"P" => agency port code<br>"B" => agency buyer(processor) code |
| AGENCY-CODE | 4-10 | alpha(7); a valid agency code |
| PACFIN-CODE | 11-14 | alpha; a valid PacFIN species, area, gear, or port code; if CODE-TYPE = "B" then this item |

is <blank>

DESCRIPTION 15-72 text; if CODE-TYPE = "S" then scientific name and/or common name and/or general description; if CODE-TYPE = "A" then description of area; if CODE-TYPE = "G" then description of gear; if CODE-TYPE = "P" then name of port; if CODE-TYPE = "B" then name of buyer

Each CODE-LIST transaction must be unique for: CODE-TYPE,AGENCY-CODE

**Fish-Ticket Specification**

| DATA ITEM | COLS | DESCRIPTION; RANGE OF VALUES |
|---|---|---|
| TRANS-ID | 1 | set to "F" => FISH-TICKET transaction |
| TRANS-OP | 2 | "+" => add, "*" => change, "-" => delete; "&" => change-or-add |
| TICKET-ID | 3-10 | alpha; any combination of letters and numbers |
| YEAR | 11-12 | integer; year of landing (e.g. 81,82,...) |
| MONTH | 13-14 | integer; month of landing (e.g. 01,02,...,12) |
| DAY | 15-16 | integer; day of landing (e.g. 01,02,...,31) |
| PORT | 17-19 | alpha; actual port of landing (not a dealer); must be a valid port code contained in the reporting agency's list of port codes previously input to the PacFIN database. |
| VESSEL-USED | 20 | alpha; "T" => a vessel was used to harvest this catch; "F" => a vessel was NOT used. |
| VESSEL-ID-TYPE | 21 | alpha; Please see code list for VID-TYPE. Although codes 7 and 8 are included in the list, they are there only to give meaning to historical data. If 7 or 8 are used the transaction will be rejected. |
| VESSEL-ID | 22-29 | alpha(8); a vessel id, license #, etc.; may be NULL |

PROCESSOR-ID 30-36 alpha(7); synonyms: processor code, dealer no.; will be verified using the list of ids previously input to the PacFIN database by the reporting agency. An invalid PROCESSOR-ID will cause a warning message to be generated, but the transaction will be accepted.

PAR-GROUP 37 alpha; participant group; "C" => non-Indian commercial fisher; "I" => treaty Indian commercial fisher.

DAYS-FISHED 38-39 integer; may be NULL (i.e. unknown)

FISHERMAN-LIC 40-45 alpha(6); may be NULL (i.e. unknown)

Each FISH-TICKET transaction must be unique for: TICKET-ID, YEAR, MONTH, DAY, and PAR-GROUP

TICKET-ID is not expected to be unique for WDFW and CDFG transactions. It is expected that the combination of TICKET-ID, PAR-GROUP, YEAR will provide a unique key for WDFW transactions. For CDFG transactions there is no guarantee that the combination of TICKET-ID, YEAR, MONTH, DAY will provide a unique identification, but the probability is high that it will. It is expected that TICKET-ID will provide a unique identifier for all ODFW transactions independent of PAR-GROUP, YEAR, MONTH, DAY. This statement is true when considering all data years, currently 1981 thru 2003.

**Fish-ticket Line Specification**

| DATA ITEM | COLS | DESCRIPTION; RANGE OF VALUES |
|---|---|---|
| TRANS-ID | 1 | set to "L" => FISH-TICKET-LINE transaction |
| TRANS-OP | 2 | "+" => add, "*" => change, "-" => delete |
| TICKET-ID | 3-10 | alpha(8); fish ticket identifier |
| TICKET-CATEGORY | 11-14 | alpha(4); synonyms: species code, market category; must be a valid code contained in the reporting agency's list of ticket-categories |

previously input to the PacFIN database.

GRADE 15 alpha(1); see code list for GRADE

CONDITION 16 alpha(1); condition of fish at time of landing
see code list for CONDITION
**********
* NOTE: transactions with condition = E (roe) will
* not be expanded to a round-weight equivalent nor
* added to round-weight summaries maintained within
* the database, since doing so would potentially
* count the same biomass (i.e. fish) twice.
**********

DISPOSITION 17 alpha(1); indicates intended use of marine species
see code list for DISPOSITION

AREA-TYPE 18 integer(1); see code list for AREA-TYPE

CATCH-AREA 19-22 alpha(4); must be a valid area code
contained in the reporting agency's list of
area codes previously input to the PacFIN
database

GEAR 23-25 alpha; gear used to harvest the catch of
marine species; must be a valid gear code contained
in the reporting agency's list of gear codes
previously input to the PacFIN database.

WEIGHT-OF-CATCH 26-34 9-digit integer; units are pounds; actual landed weight

CONVERT-FACTOR 35-39 5-digit integer; units are thousandths; this is
the factor used to convert landed weight to round
weight equivalent; must be greater than or equal
to 01000 (12345 => conversion factor of 12.345).
May be NULL.

NUMBER-OF-FISH 40-45 max 6-digit real number; number of fish caught; nominally
for salmon but to be provided where possible; may
be NULL (examples: 1234.5, 987654). This metric may be the

result of a statistical estimation process.

PRICE-PER-POUND 46-52 integer(7); units are .001 dollars per pound;
( 0001234 => $1.234 ); 0000000 is valid; may be NULL

PAR-GROUP 53 alpha; participant group; "C" => non-Indian
commercial fisher; "I" => treaty Indian commercial
fisher.
*** REQUIRED SINCE TREATY INDIANS POTENTIALLY MAY
*** USE SAME TICKET NUMBERS AS WDFW

YEAR 54-55 integer; year of landing (e.g. 81,82,...)
*** REQUIRED SINCE WDFW WILL RE-CYCLE TICKET
*** NUMBERS IN APPROXIMATELY SIX YEARS (as of 1990)

MONTH 56-57 integer; month of landing (e.g. 01,02,...,12)

DAY 58-59 integer; day of landing (e.g. 01,02,...,31)
*** YEAR,MONTH,DAY ARE REQUIRED IN THIS
*** TRANSACTION SINCE SOME CDFG TICKET-IDs
*** WERE INADVERTENTLY REUSED DURING THE
*** SAME ANNUAL PERIOD

CDFG-ORIGIN 60-63 integer(4); CDFG area of catch; for use by
CDFG only; ignored for all other data sources;
may be NULL; also known as CDFG Block Number

REMOVAL-TYPE 64 alpha; list includes: commercial, EFP, personal
use, research, commercial(direct sale). See
code list of REMOVAL-TYPE for a complete list
and description for each code

LEGAL-REMOVAL 65 alpha; 'T' => legal removal (fishing)
'F' => illegal fishing

PRODUCT-USE 66 alpha; list includes: animal food, bait, human
food, aquarium, reduction, discard, medical
research. See PRODUCT-USE code list for a
complete list and description of each code.

PRODUCT-FORM 67 alpha; list includes: fresh/frozen/dead, canned, live, meal/dried. See PRODUCT-FORM code list for a complete list and description of each code

OVERAGE 68 alpha; 'T' => true (catch over some limit) 'F' => no overage

ASSIGNMENT 69 alpha; list includes: vessel, state agency, NOAA. See ASSIGNMENT code list for a complete list and description of each code.

NWR-LE-PERMID 70-75 alpha(6); NMFS/NWR limited-entry permit identifier;

Each FISH-TICKET-LINE transaction must be unique for: TICKET-ID, YEAR, MONTH, DAY, PAR-GROUP, TICKET-CATEGORY, GRADE, CONDITION, DISPOSITION, AREA-TYPE, CATCH-AREA, GEAR, PRICE-PER-POUND, CONVERT-FACTOR, REMOVAL-TYPE, LEGAL-REMOVAL, PRODUCT-USE, PRODUCT-FORM, OVERAGE, ASSIGNMENT.

VALUES FOR COMP-TYPE:

"C" => specifies that PROPORTION is to be used with aggregates for groundfish species where the AREA on the FISH-LINE-TICKET is a coastal (ocean) area

"P" => specifies that PROPORTION is to be used with aggregates for groundfish species where the AREA on the FISH-TICKET-LINE is a Puget Sound area

"G" => specifies that PROPORTION is to be used with aggregates for groundfish species where the AREA on the FISH-TICKET-LINE is unknown

"A" => specifies that PROPORTION is to be used with aggregates for salmon species where AREA on the FISH-TICKET-LINE is unknown and trip type (DAYS-FISHED on FISH-TICKET) is known.

"T" => specifies that PROPORTION is to be used with aggregates for salmon

species where trip type is unknown and AREA is known.

"U" => specifies that PROPORTION is to be used with aggregates for salmon species where trip type and AREA are both unknown

VALUES FOR TRIP-TYPE:

"D" => specifies that PROPORTION is to be applied to day trip (DAYS-FISHED =1) aggregates

"T" => specifies that PROPORTION is to be applied to trip (DAYS-FISHED > 1) aggregates

NOTES:

Groundfish COMP-TYPES are: C, P, G

Salmon COMP-TYPES area: A, T, U

TRIP-TYPE applies to salmon only

**Area Comp Specification**

| DATA ITEM | COLS | DESCRIPTION; RANGE OF VALUES |
|---|---|---|
| TRANS-ID | 1 | set to "A" => AREA-COMP transaction |
| TRANS-OP | 2 | "+" => add, "*" => change, "-" => delete |
| PERIOD | 3 | "M" => monthly, "W" => weekly, "Q" => quarterly |
| YEAR | 4-5 | integer; year for which PROPORTION applies |
| MONTH | 6-7 | integer; month or quarter for which PROPORTION applies; if PERIOD = "Q" then specifies last month of quarter (03 => Q1, 06 => Q2, 09 => Q3, 12 => Q4) |
| DAY | 8-9 | integer; ignored if PERIOD = "M" or PERIOD = "Q"; if PERIOD="W" specifies that PROPORTION applies to the 7-day period ending on this YEAR-MONTH-DAY |

COMP-TYPE 10 alpha; type of composition value contained in item PROPORTION; see values for COMP-TYPE

PORT 11-13 alpha(3); PacFIN port code; may be NULL.

GEAR 14-16 alpha; PacFIN gear or gear-group code (e.g. "GFT", "TWL", "TRL")

PACFIN-CATEGORY 17-20 alpha; species or species group being apportioned must be a valid PacFIN species code (e.g. SABL, URCK, CHNK, COHO, PINK, SOCK, CHUM)

TRADE 21 alpha; same as in FISH-TICKET-LINE transaction; may be NULL

TRIP-TYPE 22 alpha; see VALUES FOR TRIP-TYPE; may be NULL

CATCH-AREA 23-25 alpha; a PacFIN PMFC or salmon area code

PROPORTION 26-30 5-digit integer; (01234 => 0.1234, 10000 => 1.0000) estimated PROPORTION for CATCH-AREA for the time-period,COMP-TYPE,PORT,GEAR,PACFIN-CATEGORY, GRADE,TRIP-TYPE specified

COEFFVAR 31-33 integer; (001 => 0.01, 100 => 1.00); coefficient of variance; this item is intended to document the quality of the estimate provided in item PROPORTION; optional.

NUM-SAMPLES 34-37 integer(4); number of samples used to develop PROPORTION.

Each AREA-COMP transaction must be unique for:

PERIOD,YEAR,MONTH,DAY,COMP-TYPE,PORT,GEAR,PACFIN-CATEGORY, GRADE,TRIP-TYPE,CATCH-AREA

**Species Comp Specification**

| DATA ITEM | COLS | DESCRIPTION; RANGE OF VALUES |
|---|---|---|
| TRANS-ID | 1 | set to "S" => SPECIES-COMP transaction |
| TRANS-OP | 2 | "+" => add, "*" => change, "-" => delete |
| PERIOD | 3 | "M" => monthly, "W" => weekly, "Q" => quarterly |
| YEAR | 4-5 | integer; year for which PROPORTION applies |
| MONTH | 6-7 | integer; month or quarter for which PROPORTION applies; if PERIOD = "Q" then specifies last month of quarter (03 => Q1, 06 => Q2, 09 => Q3, 12 => Q4); |
| DAY | 8-9 | integer; ignored if PERIOD = "M" or PERIOD = "Q"; if PERIOD="W" specifies that PROPORTION applies to the 7-day period ending on this YEAR-MONTH-DAY |
| PORT | 10-12 | alpha; a PacFIN port code unless data source is CDFG, then a sample port group is required; may be NULL; NULL => no stratification by port |
| AREA | 13-15 | alpha; must be a PacFIN PSMFC area or subarea code; may be NULL; NULL => no stratification by area |
| GEAR | 16-18 | alpha; a PacFIN gear or gear-group code; (e.g. "GFT", "TWL") |
| UNSP-SPECIES | 19-22 | alpha(4); a PacFIN unspecified species code or market category. Example of codes allowed: "LSP1" => nominal longspine thornyhead "CNR1" => nominal canary rockfish See the PacFIN species list for a complete list of available codes. |
| SPECIES | 23-26 | alpha; a PacFIN rockfish species (e.g. "CNRY") |
| PROPORTION | 27-31 | 5-digit integer; (01234 => 0.1234, 10000 => 1.0000) |

estimated PROPORTION for SPECIES within the specified time-period,PORT,AREA,GEAR,UNSP-SPECIES stratum

COEFFVAR 32-34 integer; (001 => 0.01, 100 => 1.00); coefficient of variance; this item is intended to document the quality of the estimate provided in item PROPORTION; optional.

NUM-SAMPLES 35-38 integer(4); number of samples used to develop PROPORTION.

SAMPLE-COND 39-39 alpha; condition of fish at time of sampling; at this time one of three values is allowed:
"L" = live, "D" = dead, or "9" = all landing conditions

Each SPECIES-COMP transaction must be unique for:

YEAR,MONTH,DAY,PORT,AREA,GEAR,UNSP-SPECIES,SAMPLE-COND,SPECIES

**Average Weight Specification**

| DATA ITEM | COLS | DESCRIPTION; RANGE OF VALUES |
|---|---|---|
| TRANS-ID | 1 | set to "W" => AVERAGE-WEIGHT transaction |
| TRANS-OP | 2 | "+" => add, "*" => change, "-" => delete |
| PERIOD | 3 | "D"=daily, "M"=monthly, "W"=weekly, "Q"=quarterly |
| YEAR | 4-5 | integer; year for which PROPORTION applies |
| MONTH | 6-7 | integer; month or quarter for which PROPORTION applies; if PERIOD = "Q" then specifies last month of quarter (03 => Q1, 06 => Q2, 09 => Q3, 12 => Q4); |
| DAY | 8-9 | integer; ignored if PERIOD = "M" or PERIOD = "Q"; if PERIOD="W" specifies that PROPORTION applies to the 7-day period ending on this YEAR-MONTH-DAY |

PORT 10-12 alpha; a PacFIN port code; may be NULL;
NULL => no stratification by port

AREA 13-15 alpha; must be a PacFIN area code;
may be NULL; NULL => no stratification by area

GEAR 16-18 alpha; must be a valid PacFIN gear code
(e.g. TRL)

SPECIES 19-22 alpha(4); must be a valid PacFIN species code;
only codes currently allowed: CHNK, COHO, PINK,
CHUM, SOCK, STLH, SHAD, WSTG(white sturgeon),
and GSTG(green sturgeon). *** OTHERS? ***

GRADE 23 alpha; same as in FISH-TICKET-LINE transaction

CONDITION 24 alpha; same as in FISH-TICKET-LINE transaction

AVERAGE-WT 25-29 5-digit integer; units are .001 pounds per fish;
( 12345 => 12.345 ); average landed weight per fish

COEFFVAR 30-32 integer; (001 => 0.01, 100 => 1.00); coefficient of
variance; this item is intended to document the
quality of the estimate provided in item PROPORTION;
optional.

NUM-SAMPLES 33-36 integer(4); number of samples used to develop
this average landed weight.

Each AVERAGE-WEIGHT transaction must be unique for:

PERIOD,YEAR,MONTH,DAY,PORT,AREA,GEAR,SPECIES,GRADE,CONDITION

**Effort Comp Specification**

| DATA ITEM | COLS | DESCRIPTION; RANGE OF VALUES |
|---|---|---|
| TRANS-ID | 1 | set to "E" => EFFORT-COMP transaction |
| TRANS-OP | 2 | "+" => add, "*" => change, "-" => delete |
| PERIOD | 3 | "M" => monthly, "W" => weekly, "Q" => quarterly |
| YEAR | 4-5 | integer; year for which PROPORTION applies |
| MONTH | 6-7 | integer; month or quarter for which PROPORTION applies; if PERIOD = "Q" then specifies last month of quarter (03 => Q1, 06 => Q2, 09 => Q3, 12 => Q4); |
| DAY | 8-9 | integer; ignored if PERIOD = "M" or PERIOD = "Q"; if PERIOD="W" specifies that PROPORTION applies to the 7-day period ending on this YEAR-MONTH-DAY |
| COMP-TYPE | 10 | alpha; type of composition value contained in item PROPORTION; see VALUES for COMP-TYPE |
| PORT | 11-13 | alpha; a PacFIN port code; may be NULL; NULL => no stratification by port |
| GEAR | 14-16 | alpha; a PacFIN gear code; only value currently allowed is "TRL" |
| MANAGEMENT-GRP | 17-20 | alpha; a PacFIN management group code; only values currently allowed are "SAMN" and "GRND" |
| TRIP-TYPE | 21 | alpha; see TRIP-TYPE values; may be NULL |
| CATCH-AREA | 22-24 | alpha; a PacFIN salmon or groundfish area code |
| EFFORT-TYPE | 25 | alpha; "D" => deliveries "T" => time (days fished) - salmon only |
| PROPORTION | 26-30 | 5-digit integer; (01234 => 0.1234) |

COEFFVAR 31-33 integer; (001 => 0.01, 100 => 1.00); coefficient of variance; this item is intended to document the quality of the estimate provided in item PROPORTION; optional.

NUM-SAMPLES 34-37 integer(4); number of samples used to develop PROPORTION.

Each EFFORT-COMP transaction must be unique for:

PERIOD,YEAR,MONTH,DAY,PORT,GEAR,MANAGEMENT-GRP,COMP-TYPE,TRIP-TYPE,
CATCH-AREA,EFFORT-TYPE

**Vessel Specification**

| DATA ITEM | COLS | DESCRIPTION; RANGE OF VALUES |
|---|---|---|
| TRANS-ID | 1 | set to "V" => STATE-VESSEL transaction |
| TRANS-OP | 2 | "+" => add, "*" => change, "-" => delete |
| VESSEL-ID-TYPE | 3 | alpha(1);<br>1 => USCG documentation number<br>2 => State Marine Board number<br>5 => Canadian Vessel<br>U => unknown (i.e. VESSEL-ID is empty) |
| VESSEL-ID | 4-11 | alpha(8); a USCG document number (e.g. 1234567) or a USCG mandated small vessel identifier (e.g. OR123XYZ) |
| REG-NUMBER | 12-19 | alpha(8); registration (plate) number issued to vessel by state fishery agency; may be NULL<br><br>Note: either VESSEL-ID or PLATE-NUMBER must be provided |
| LENGTH-TYPE | 20 | integer(1); type of measurement used for reporting vessel length: 1 => overall length in feet;<br>2 => keel length in feet; 3 => USCG length in feet |
| VESSEL-LENGTH | 21-23 | integer(3); units=feet; length of vessel |
| WEIGHT-TYPE | 24 | integer(1); type of measurement used for reporting weight: 1 => net weight in short tons<br>2 => gross weight in short tons |
| WEIGHT | 25-29 | integer(5); weight of vessel; units = short tons |
| HORSEPOWER | 30-33 | integer(4); horsepower of main engine as reported by owner or USCG |
| YEAR | 34-35 | integer(2); designates the state agency's annual data set from which this data is derived. 97 => 1997;<br>00 => 2000; 01 => 2001 |

VESSEL-NAME 36-72 text; 37 characters max

OWNER 73-104 alpha(32); text; owner's last name followed by first name(s) or company name. The current, or last, owner for the calendar year will be provided.

STREET 105-139 alpha(35); text; owner's street address

CITY 140-156 alpha(17); text; owner's city

STATE 157-158 alpha(2); two character state code

ZIPCODE 159-168 alpha(10); five or nine digit zip code

CHARTER-BOAT 169-169 alpha(1); boolean: 'T' => used as a commercial charter boat at any time during year; 'F' => not used as a charter boat during the license YEAR specified

For CDFG and WDFW and ADFG, each STATE-VESSEL transaction must be unique for:
PLATE-NUMBER

For ODFW each STATE-VESSEL transaction must be unique for: VESSEL-ID

**Aggregated Catch Specification**

| DATA ITEM | COLS. | RANGE OF VALUES; DESCRIPTION |
|---|---|---|
| TRANS-TYPE | 1 | see definition for TRANS-OP and TRANS-TYPE<br>"&" => change_or_add, "-" => delete |
| INPUT-AGG | 2 | Input Aggregation Level<br>values: D (DAILY); CDFG, ODFW, and WDFW only<br>W (WEEKLY); NMFS/AFSC and NMFS/AKR only<br>M (MONTHLY); ADFG and DFO only<br>Z (pseudo-month: 28-35 days); NWAFC only |
| YEAR | 3-4 | year of landing; values: 81, 82, 83, ... etc. |
| MONTH | 5-6 | month of landing; values: 01-12 |

DAY 7-8 day or week of landing; values: 01-31; if INPUT-AGG is WEEKLY this item identifies a Sunday thru Saturday week where Saturday is the day specified. Will be set to zero if INPUT-AGG is not DAILY OR WEEKLY.

SPECIES 9-12 must be a valid species or species category; verified using the set of SPECIES-IDs contained in the species list; this transaction is rejected if invalid.

AREA 13-14 If equal to "**" then the area is found in columns 43-45, otherwise must be a valid PMFC Groundfish or Shrimp area, an INPFC area, or a PacFIN Salmon area, depending on the species code contained in item SPECIES

PORT-CNTRY 15-17 must be a valid PCID code: PORT (CDFG,ODFW,WDFW), SUB-REGION (ADFG), or AT-SEA CODE (AKR,AFSC,NWR); Verified using the set of PCIDs for each agency. See PSMFC/PacFIN website for a list of all port, country, sub-region, and at-sea codes. Rejected if invalid.

GEAR 18-20 must be a valid level 1 GRID code; verified using the set of GRIDs contained in PacFIN table GR; see PSMFC/PacFIN website for a list of all level 1 gear codes; rejected if invalid

WEIGHT 21-29 9-digit integer right justified, zero or blank fill; units are pounds (state agencies only) or .01 metric tons (NWR, AFSC, AKR, DFO only), required if TRANS-TYPE is CHANGE-or-ADD, ROUND WEIGHT EQUIVALENT only

NUM-LNDGS 30-33 4-digit integer right justified, zero or blank fill; number of landings for this catch; ignored if TRANS-TYPE is "-" or SOURCE is "NWAFC"; number of landings (fish tickets) from which this trans-action is derived

NUM-FISH 34-39 6-digit integer right justified, zero or blank fill; number of fish for this catch; ignored if

TRANS-TYPE is "-" or SOURCE is "AFSC"; optional

GRADE/SIZE 40 "X" => extra small; "S" => small; "M" => medium; "L" => large; "U" or " " => unspecified (mixed sizes);

FTL-AGG 41 Internal PacFIN use only; If equal to "Y" then this transaction was generated by the PacFIN system. If not equal to "Y" then this transaction was generated by a data source agency.

REVENUE-FLAG 42 Internal PacFIN use only; If = "$" then the units for the VAL-PRICED field in this transaction is dollars ($US); If not = "$" then units for VAL-PRICED is cents.

AREA-3 43-45 If data item AREA (Col. 13-14) is equal to "**" then this field contains the area-id. If a two character are is used then the last character must be blank (i.e. left justified over blank fill);

TRIP-TYPE 46 D = day trip; T = multi-day trip; U = unspecified See list of trip types in Oracle table cl.

47-51 *NOT USED*

LBS-PRICED 52-60 9 digit integer right justified, zero or blank fill; units are same as units for WEIGHT; submitted by state(province) agencies only; this item will contain the total number of units within this aggregation that were actually priced. This item may not be greater than the integer contained in data item WEIGHT and if VAL-PRICED is zero then this item must be zero also.

VAL-PRICED 61-68 8 digit integer right justified, zero or blank fill; submitted by state agencies only; this item will contain the $ value of the pounds, within this aggregation that were actually priced. If LBS-PRICED is zero then this item must be zero also. The units are dollars if SOURCE is ADFG, otherwise units are cents.

CONDITION 69 "R" => round; "D" => dressed; "F" => fileted;
"Y" => frozen-round; "Z" => frozen-dressed
"U" or " " => unspecified; REQUIRED; See Oracle table cl

HEADS 70 "Y" => heads on; "N" => heads off; 'U' => unspecified
See code-list HEADS in Oracle table cl

PAR-GRP 71 participant group; values: "C" => commercial non-Indian; "I" => Indian commercial (treaty Indian);
'U' => unspecified. See PAR-GROUP in Oracle table cl.

TRANS-ID 72 transaction identification; set to "C" for an AGGREGATED-CATCH transaction

When aggregating deliveries, be sure to count each landing receipt for a particular management group once and only once, even if more than one category of fish is recorded on the receipt. The criteria for counting a landing receipt as a delivery is:

1.) If the landing receipt contains any salmon data then it is counted as a salmon delivery;

2.) If the landing receipt is used to generate one or more aggregated catch transactions then it is counted as one delivery if the majority of the catch (in pounds) is either groundfish or pink shrimp and no salmon catch was recorded;

3.) Landing receipts where the majority of the catch is Pacific Halibut and/or California Halibut will not be counted as a groundfish delivery;

4.) Deliveries for Pacific Halibut and California Halibut will not be submitted;

5.) A 50-50 split between Groundfish and Pink Shrimp will be classified as a Groundfish delivery.

**Aggregated Effort Specification**

| DATA ITEM | COLS. | RANGE OF VALUES; DESCRIPTION |
|---|---|---|
| TRANS-TYPE | 1 | "&" or "-"; see above description |
| INPUT-AGG | 2 | Input Aggregation Level<br>values: D (DAILY); CDFG, ODFW, and WDFW only<br>W (WEEKLY); NMFS/NWAFC and NMFS/AKR only<br>M (MONTHLY); ADFG, CDFG, DFO, ODFW, and WDFW only<br>Z (pseudo-month: 28-35 days); NWAFC only |
| YEAR | 3-4 | year of landing; values: 81, 82, 83, ... etc. |
| MONTH | 5-6 | month of landing; values: 01-12 |
| DAY | 7-8 | day or week of landing; values: 01-31; if INPUT-AGG is WEEKLY this item identifies a Sunday thru Saturday week where Saturday is the day specified. Will be set to zero if INPUT-AGG is not DAILY OR WEEKLY. |
| MNG-GRP | 9-12 | a PacFIN management group; values: "GRND" => groundfish (excluding halibut); "PSHP" => pink shrimp; and "SAMN" => salmon |
| AREA | 13-14 | If equal to "**" then the area is found in columns 41-43, otherwise must be a valid PMFC Groundfish or Shrimp area, an INPFC area, or a PacFIN Salmon area, depending on the management group contained in item MNG-GRP |
| PORT | 15-17 | must be a valid PORT; only state and provincial agencies may submit this transaction; verified using the set of PORT-IDs contained in the port list for the reporting agency |
| GEAR | 18-20 | must be a valid GEAR; verified using the set of GEAR-IDs contained in GEAR-LIST; rejected if invalid |

DELIVERIES 21-26 6 digit integer right justified; zero or blank fill; aggregated number of deliveries for the day(or month) of landing, management group, area, gear, and port specified within this transaction; required if TRANS-TYPE is CHANGE-or-ADD

DAYSFISHED 27-32 6 digit integer right justified; zero or blank fill; aggregated number of days-fished for the day(or month) of landing, management group, area, gear, and port specified within this transaction; optional; units are .1 days

TRAWL-HRS 33-40 8 digit integer right justified; zero or blank fill; aggregated number of trawl hours of fishing for the catch landed on a day or within a month for a particular MNG-GRP, AREA, PORT, and GEAR; only applies to trawl gears (including shrimp trawls); ignored if gear is non-trawl; units are .1 hours

FTL-AGG 41 Internal PacFIN use only. This field is set by the UPD/PACFIN program.

AREA-3 42-44 If data item AREA (Col. 13-14) is equal to "**" then this field contains the area-id. If a two character are is used then the last character must be blank (i.e. left justified over with blank fill).

TRIP-TYPE 45 D = day trip; T = multi-day trip; U = unspecified See list of trip types in Oracle table cl.

46-70 *NOT USED*

PAR-GRP 71 participant group; values: "C" => commercial non-Indian; "I" => Indian commercial (treaty Indian)

TRANS-ID 72 transaction identification; set to "D" for an AGGREGATED-EFFORT transaction

**WDFW Logbook Data Feed Specification - 11/20/2010**

DESCRIPTION:
THIS FORMAT WILL BE USED FOR EXPORTING TRAWL LOGBOOK DATA FROM THE
WDFW LOGBOOK TABLES IN SQL SERVER TO A TEXT FILE FOR TRANSMITTAL TO
PACFIN. FOUR DIFFERENT RECORD TYPES (TRIP, TOW, CATCH, AND LBK-TKT)
ARE WRITTEN TO THIS FILE. TRIP ID IS AN INTEGER NUMBER THAT UNIQUELY
IDENTIFIES EACH TRIP. TOW NUMBER IS A SEQUENTIAL NUMBER FOR EACH TOW.
TRIP TYPES ARE: 1=CANADA, 2=ALASKA, 3=COASTAL, 4=STRAITS & GULF,
5=PUGET SOUND & HOOD CANAL. LATITUDE AND LONGITUDE ARE IN DECIMAL
DEGREES TO FOUR DECIMAL PLACES WITH AN IMPLIED DECIMAL POINT
(E.G., 1225327 = 122.5327).

FORMAT OF RECORD TYPE 1 (TRIP)

| FIELD | COLUMNS | DESCRIPTION | SIZE | VALUE |
|---|---|---|---|---|
| 1 | 1-1 | RECORD TYPE | 1 | 1 |
| 2 | 2-13 | TRIP ID | 12 | |
| 3 | 14-19 | VESSEL ID | 6 | |
| 4 | 20-21 | DEPART YEAR | 2 | |
| 5 | 22-23 | DEPART MONTH | 2 | |
| 6 | 24-25 | DEPART DAY | 2 | |
| 7 | 26-29 | DEPART TIME | 4 | |
| 8 | 30-32 | DEPART PORT | 3 | |
| 9 | 33-34 | RETURN YEAR | 2 | |
| 10 | 35-36 | RETURN MONTH | 2 | |
| 11 | 37-38 | RETURN DAY | 2 | |
| 12 | 39-42 | RETURN TIME | 4 | |
| 13 | 43-45 | RETURN PORT | 3 | |
| 14 | 46-47 | DAYS OUT | 2 | |
| 15 | 48-49 | DAYS FISHED | 2 | |
| 16 | 50-51 | CREW SIZE | 2 | |
| 17 | 52-52 | TRIP TYPE | 1 | |
| 18 | 53-54 | NO. OF TOWS | 2 | |

FORMAT OF RECORD TYPE 2 (TOW)

| FIELD | COLUMNS | DESCRIPTION | SIZE | VALUE |
|---|---|---|---|---|
| 1 | 1-1 | RECORD TYPE | 1 | 2 |
| 2 | 2-13 | TRIP ID | 12 | |
| 3 | 14-15 | TOW NUMBER | 2 | |
| 4 | 16-17 | TOW YEAR | 2 | |
| 5 | 18-19 | TOW MONTH | 2 | |
| 6 | 20-21 | TOW DAY | 2 | |
| 7 | 22-25 | SET TIME | 4 | |
| 8 | 26-29 | UP TIME | 4 | |
| 9 | 30-31 | TOW HOURS | 2 | |
| 10 | 32-33 | TOW MINUTES | 2 | |
| 11 | 34-39 | SET LATITUDE | 6 | |
| 12 | 40-46 | SET LONGITUDE | 7 | |
| 13 | 47-52 | UP LATITUDE | 6 | |
| 14 | 53-59 | UP LONGITUDE | 7 | |
| 15 | 60-61 | WDFW AREA | 2 | |
| 16 | 62-63 | MF MGMT AREA | 2 | |
| 17 | 64-65 | GROUND CODE | 2 | |
| 18 | 66-67 | PSMFC AREA | 2 | |
| 19 | 68-71 | AVG DEPTH | 4 | |
| 20 | 72-72 | NET TYPE | 1 | |
| 21 | 73-76 | TARGET STRATEGY | 4 | |

FORMAT OF RECORD TYPE 3 (CATCH)

| FIELD | COLUMNS | DESCRIPTION | SIZE | VALUE |
|---|---|---|---|---|
| 1 | 1-1 | RECORD TYPE | 1 | 3 |
| 2 | 2-13 | TRIP ID | 12 | |
| 3 | 14-15 | TOW NUMBER | 2 | |
| 4 | 16-18 | SPECIES | 3 | |
| 5 | 19-24 | HAILED LBS. | 6 | |

FORMAT OF RECORD TYPE 4 (LBK-TKT)

| FIELD | COLUMNS | DESCRIPTION | SIZE | VALUE |
|---|---|---|---|---|
| 1 | 1-1 | RECORD TYPE | 1 | 4 |
| 2 | 2-13 | TRIP ID | 12 | |
| 3 | 14-20 | TICKET NUMBER | 7 | |

THE FOLLOWING NON-NULL FIELDS UNIQUELY IDENTIFY ROWS FOR EACH RECORD TYPE:

| | |
|---|---|
| TRIP | RECORD TYPE, TRIP ID |
| TOW | RECORD TYPE, TRIP ID, TOW NUMBER |
| CATCH | RECORD TYPE, TRIP ID, TOW NUMBER, SPECIES |
| LBK-TKT | RECORD TYPE, TRIP ID, TICKET NUMBER |

ROWS WILL BE GROUPED BY TRIP, RECORD TYPE 1 AHEAD OF RELATED ROWS FOR RECORD TYPE 2, FOLLOWED BY RELATED ROWS FOR RECORD TYPE 3, ETC.

SOURCE INPUT SPECIFICATION
==========

All Coast-Wide Biological data will be input using the following transactions (or record types):

1. SAMPLE Transaction (trans_id="G")
2. CLUSTER Transaction (trans_id="H")
3. FISH Transaction (trans_id="I")
4. AGE Transaction (trans_id="J")
5. AGENCY-CODE-LIST Transaction (trans_id="K")

All data items will be comma-delimited, without any padding or justification. Missing or null data items will be input as an empty string (the leading comma will be followed by the trailing comma with no characters in between).

All data items are either alpha, text, integer, or number. Alpha items are uppercase letters and/or numbers. Text data items may contain any printable character including the <blank> and must be enclosed in quotation marks if the item contains a comma. Integer items are numbers only. Number items are numbers with an optional decimal point.

All data items are optional unless specified otherwise.

A Coast-Wide Biological datafeed will consist of a data file which will include a single report record (must be the first record in the file) and one or more transaction records. This data file need not be sorted in any particular order, except that the report record must be the first record. Any sorting that may be necessary will be accomplished by the central processing system.

Datafeeds will be provided by each source on an annual basis. The minimum requirement for each source is an annual datafeed of prior year data by April of the following year, but updates can be provided as often as desired between annual submissions. Updates will be processed on a full-year refresh basis, with the update datafeed completely replacing any previous annual datafeed from that source.

In the following record and transaction specifications, the SIZE column

indicates the maximum size for that data item and the DESCRIPTION column indicates the data type and details of which sources will be providing that data item. A source is identified by a one-character PacFIN agency code (agid) inside square brackets. When a description applies to more than one source, there will be multiple agency codes concatenated together inside the brackets. The PacFIN agency codes are defined as follows:

| AGID | AGENCY NAME |
|---|---|
| W | Washington Dept of Fish and Wildlife |
| O | Oregon Dept of Fish and Wildlife |
| C | California Dept of Fish and Game |
| M | Northwest Fisheries Science Center |
| F | Alaska Fisheries Science Center |

Ordering of data items within each type of transaction is as follows:

1. Data items making up the primary key (all of which are required).
2. Data items that are common to multiple sources and single source data items except for Oregon, ordered alphabetically.
3. Oregon-only data items, ordered alphabetically.

The reason the Oregon-only data items are listed at the end of each transaction is to make it easy to identify them and to allow the other sources to submit shorter transaction records.

All unit of measure (um) elements will be converted to common units in the coast-wide database.

AGENCY-CODE-LIST (ACL) transactions differ from the other types of transactions in that they are not tied to a particular sample year. Because of this, ACL transactions are processed as add-or-update transactions instead of the annual refresh method used for the other types of transactions. The implication of this is that a given agency code value can only have one meaning for all years of data. In other words, an agency may not use the same code to mean different things in different years.

The report record consists of the following data items:

DATA ITEM SIZE DESCRIPTION; RANGE OF VALUES

---

source 5 Alpha. Required.
Agency acronym.
[WOCMF] Values:
WDFW = Washington Dept. of Fish and Wildlife,
ODFW = Oregon Dept. of Fish and Wildlife,
CDFG = California Dept. of Fish and Game,
NWFSC = Northwest Fisheries Science Center,
AFSC = Alaska Fisheries Science Center.

rpt_date 8 Integer. Required.
[WOCMF] YYYYMMDD (e.g. 19981012); this is the date of report; (i.e. the date the datafeed was generated); two datafeeds from the same source may not have the same date.

comm_desc 59 Text.
[WOCMF] Comment or description regarding this periodic report.

The AGENCY-CODE-LIST transaction consists of the following data items or attributes:

DATA ITEM SIZE DESCRIPTION; RANGE OF VALUES
---

trans_id 1 Alpha. Required.
[WOCMF] Set to "K" => AGENCY-CODE-LIST transaction.

code_type 1 Alpha. Required.
The type of agency code being submitted.
[WOCMF] Values:
S = agency species code,
A = agency area code,
G = agency gear code,
P = agency port code.

agency_code 7 Alpha. Required.
[WOCMF] A valid agency code.

pacfin_code 4 Alpha. Required.

[WOCMF] A valid PacFIN species, area, gear or port
code.

description 58 Text. Required.
[WOCMF] If code_type = S then scientific name and/or
common name and/or general description;
if code_type = A then description of area;
if code_type = G then description of gear;
if code_type = P then name of port.

Each AGENCY-CODE-LIST transaction must be unique for: code_type, agency_code

The SAMPLE transaction consists of the following data items or attributes:

| DATA ITEM | SIZE | DESCRIPTION; RANGE OF VALUES |
|---|---|---|
| trans_id | 1 | Alpha. Required.<br>[WOCMF] Set to "G" => SAMPLE transaction. |
| sample_no | 14 | Alpha. Required.<br>Unique number for each sample.<br>[W] sample year + species id + sequence number.<br>[O] sample_no.<br>[C] sample year + port code + quarter + sample number.<br>[M] sampyr + sampnum.<br>[F] year + portcode + sampletype + samplercode + samplenumber. |
| block | 5 | Alpha.<br>[O] block; ODFW fishing area block number.<br>[C] CDFG block number. |
| comments | 80 | Text.<br>[W] comments. |
| cond_agcode | 3 | Alpha.<br>Transferred to the coast-wide database as agency codes, then converted to PacFIN condition codes.<br>[O] condition;<br>Values: |

0 = round,
1 = dressed - head on,
2 = dressed - head off,
3 = dressed - head/tail off,
4 = shucked,
5 = frozen - dressed - head on,
6 = frozen - dressed - head off,
7 = frozen - round,
8 = filleted,
9 = live.

[M] cond;
Values:
D = dressed,
W = whole.

[F] mainproduct;
Values:
SUR = surimi,
H&G = headed & gutted,
FIL = fillets.

data_type 1 Alpha.
[WOCMF] "C" (constant).

dealer 29 Text.
[O] dealer; ODFW dealer code.
[F] plantname; Name of plant where sample was obtained.

depth_avg 8 Integer.
[W] depth fished; fathoms.
[O] depth; Average depth that most fish were caught
(default is depth of tow/haul with most fish).
fathoms.
[C] depth; fathoms.

depth_max 8 Integer.
[O] max_depth; Maximum depth fished. fathoms.
[M] depthmax; fathoms.

depth_min 8 Integer.
[O] min_depth; Minimum depth fished. fathoms.

[M] depthmin; fathoms.

disp_agcode 2 Alpha.
Transferred to the coast-wide database as agency codes, then converted to PacFIN disposition codes.
[W] fish use;
Values:
A = animal food,
B = fish bait,
C = cruise data,
D = discard,
E = (eggs) roe,
H = human food,
I = incidental catch,
K = spawn on kelp,
O = other,
R = reduction.
[O] disp_code;
Values:
01 = catch retained and landed,
02 = catch discarded at sea,
03 = pre-sorted catch,
09 = overage.

females_num 5 Integer.
[O] female_num; total number of females in sample;
[F] femalesnum; Number of females in sample.

females_wgt 9 Number (precision 8 and scale 3, example 12345.678).
[O] female_wt; weight of females in sample in pounds.
[F] femaleswtkg; weight of females in sample in kg.

females_wgt_um 1 Alpha. Required if females_wgt is present.
[OF] Unit of measure (K = kg, L = lb)

ftid 9 Alpha.
[W] fish ticket number; blank before 1998.
[O] ticket; Landing receipt (fish ticket) number
[C] landing receipt number
[M] ftid

[F] ticket

gear 5 Alpha.
Transferred to the coast-wide database as agency codes, then converted to PacFIN gear codes.
[W] gear type.
[O] gear.
[C] gear code.
[M] gear.

grade_agcode 2 Alpha.
Transferred to the coast-wide database as agency codes, then converted to PacFIN grade codes.
[O] grade; Market length or weight grade of species sampled.
Values:
0 = ocean run,
1 = extra small,
2 = small,
3 = medium,
4 = large.
[M] srtgrp;
Values:
XS = extra small,
S = small,
M = medium,
L = large,
XL = extra large,
OR = ocean run (most are not further graded).

inpfc_area 3 Alpha.
Transferred to coast-wide database as agency codes, then converted to PacFIN area codes of type 2.
[W] region; extended list of INPFC areas;
Values:
CS = Central Sound,
CHR = Charlotte, Canada,
COL = Columbia,
CON = Conception,
EUR = Eureka,

GB = Gulf-Bellingham,
HC = Hood Canal,
JF = Juan de Fuca,
MON = Monterrey,
PS = Puget Sound,
SJ = San Juans,
SS = South Sound,
SEA = Southeast Alaska,
VCN = Vancouver, Canada,
VUS = Vancouver, U.S.,
WJ = West Juan de Fuca.

[O] inpfc;
Values:
CP = Conception,
EU = Eureka,
MO = Monterey,
NC = North Columbia,
SC = South Columbia,
VN = Vancouver,
ZZ = Cobb Seamount.

[C] inpfc; derived from port code.
Values:
CON = Conception,
CP = Conception,
EK = Eureka,
EUR = Eureka,
MON = Monterey,
MT = Monterey.

[M] inpfc;
Values:
COL = Columbia,
CON = Conception,
EUR = Eureka,
MON = Monterey,
VAN = Vancouver (US).

latitude 10 Number (precision 9 and scale 5, example 1234.56789).
[O] latitude; Latitude in decimal degrees.
[C] latitude; may be added in future.

longitude 10 Number (precision 9 and scale 5, example 1234.56789).
[O] longitude; Longitude in decimal degrees.
[C] longitude; may be added in future.

males_num 5 Integer.
[O] male_num; Total number of males in sample.
[F] malesnum; number of males in sample.

males_wgt 9 Number (precision 8 and scale 3, example 12345.678).
[O] male_wt; weight of males in sample in pounds.
[F] malewtkg; weight of males in sample in kg.

males_wgt_um 1 Alpha. Required if males_wgt is present.
[OF] Unit of measure (K = kg, L = lb)

market_category 3 Alpha.
Will be converted to PacFIN codes (TBD).
[O] mkt_cat; ODFW market category of species sampled.
Values:
041 = Skate,
406 = Small rockfish (slope),
410 = Large rockfish (shelf),
413 = Pacific Ocean perch,
431 = Widow rockfish,
433 = Yellowtail rockfish,
442 = Black rockfish,
451 = Canary rockfish,
468 = Mixed thornyhead,
470 = Shortspine thornyhead,
471 = Longspine thornyhead,
477 = Sablefish,
484 = Lingcod,
606 = Arrowtooth flounder,
608 = Petrale sole,
624 = Dover sole,
626 = English sole.
[C] market category; California market category.
Values: 1 to 999.

mesh_size 2 Integer.

[W] mesh size; mesh size in tenths of inches.

other_sample_no 14 Alpha.
[W] other sample no; if received from another agency.
[M] statenum; state sample number.

port 5 Alpha.
Transferred to coast-wide database as agency codes,
then converted to PacFIN port codes.
[W] port.
[O] port.
[C] port code.
[M] port.
[F] portcode.

psmfc_area 2 Alpha.
PacFIN area codes of type 1.
[W] pmfc area;
Values: 1C, 2A, 2B, 2C, 2E, 2F, 3A, 3B, 3C.
[O] pmfc; PMFC area of catch (default is area of port
if area of catch is not known).
Values: 1C, 2A, 2B, 2C, 2E, 2F, 3A, 3B, CS.
[C] psmfc area; derived from port code.
Values: 1A, 1B, 1C.

sample_agency 4 Alpha.
The source agency that obtained the biological sample.
Transferred to the coast-wide database as agency codes,
then converted to PacFIN agency codes (agid).
[W] data source;
Values:
C = CDFG,
N = NMFS Newport,
O = ODFW,
T = NMFS Tiburon,
W = WDFW.
[O] agency;
Values:
CA = CDFG,
ED = Enhanced Data Collection Project,

MS = Mackerel Studies,
OR = ODFW,
PW = Pacific Whiting,
SP = Special Projects,
WA = WDFW.

[C] "CA" (constant).

[M] PacFIN agid of source agency; derived from NMFS sampnum;
Values:
C = CDFG,
O = ODFW,
T = Tribal (treaty indian),
W = WDFW.

[F] PacFIN agid of source agency; derived from HAKEPORT portcode.
Values:
C = CDFG,
O = ODFW,
W = WDFW.

sample_day 2 Integer.

[W] sample day.

[O] day from land_date.

[C] day from sample date.

[M] sampday.

[F] day from sampledate.

sample_method 1 Alpha.

[W] sample method;
Values:
N = systematic (every Nth fish),
P = purposive (meets certain criteria),
R = random,
S = stratified,
X = special (e.g. extrapolation).

[O] R = random (constant)

[C] R = random (constant)

[M] R = random (constant)

[F] R = random (constant)

sample_month 2 Integer.
[W] sample month.
[O] month from land_date.
[C] month from sample date.
[M] sampmon.
[F] month from sampledate.

sample_strat 1 Alpha.
[W] sample strat; if sample_method = S.
Values:
A = age,
L = length,
M = maturity,
W = weight
[F] sampletype;
Values:
1 = length frequency,
2 = otolith sample

sample_type 1 Alpha.
[W] sample type;
Values:
C = commercial on-board,
M = market sample,
R = research cruise,
S = special request.
[O] M = Market sample (constant).

sample_year 4 Integer.
[W] sample year.
[O] year from land_date.
[C] year from sample date.
[M] sampyr.
[F] year from sampledate.

sampler_code 3 Alpha.
[W] sampler initials.
[O] obsnum; ODFW assigned sampler/observer ID number.
[F] samplercode; Sampler number, sequential within a port when there is more than 1 sampler for a port.

second_stage_unit 1 Alpha.
[WOCMF] second stage of sample design;
Values:
N = fixed number
W = fixed weight

total_clusters 1 Integer.
[O] nclust; number of clusters in sample.
[M] totclust.

total_fish 4 Integer.
[W] fish count.
[O] tot_num; Total number of fish in sample.

total_wgt 10 Number (precision 9 and scale 3, example 123456.789).
[W] catch weight; pounds.
[O] land_wt; landing weight from the landing receipt in pounds.
[C] landing weight; landed weight of sampled market category in pounds.
[M] wgtland; Pounds.
[F] landingwtmt; landing weight of entire delivery in metric tons.

total_wgt_um 1 Alpha. Required if total_wgt is present.
[WOCMF] Unit of measure (L = lb, M = mtons).

veid 10 Alpha.
Will be converted to Federal codes if needed.
[W] vessel id; Federal ID.
[O] docnum; 6-7 digit Federal document number.
[C] Federal documentation number; derived from state vessel number.
[M] boatnum.
[F] boatid; Boat identification number or name.

weight_source 1 Alpha.
[W] weight source;
Values:

B = buyer's weight (fish ticket),
C = cruise data,
F = fisher's estimate,
S = sampler's estimate.
[O] B = buyer's weight (constant).

wgtmax 3 Integer.
[M] wgtmax. Top of weight range that processor is using to define srtgrp for this sample. Pounds.

wgtmin 3 Integer.
[M] wgtmin. Bottom of weight range that processor is using to define srtgrp for this sample. Pounds.

adj_bothn 3 Integer.
[O] adj_bothn; Total number of "adjusted" fish in sample.

adj_fmnum 3 Integer.
[O] adj_fmnum; Total number of females minus number of females that can't be aged.

adj_fmwt 8 Number (precision 7 and scale 1, example 123456.7).
[O] adj_fmwt; Total weight of females minus estimated weight of females that can't be aged (estimates come from the most current length/weight equation accepted for that species and sex). Pounds.

adj_mnum 3 Integer.
[O] adj_mnum; Total number of males minus number of males that can't be aged.

adj_mwt 8 Number (precision 7 and scale 1, example 123456.7).
[O] adj_mwt; Total weight of males minus estimated weight of males that can't be aged (estimates come from the most current length/weight equation accepted for that species and sex). Pounds.

chan1 2 Integer.
[O] chan1; Descriptor of 1st loran coordinate.

chan2 2 Integer.
[O] chan2; Descriptor of 2nd loran coordinate.

exp_wt 8 Integer.
[O] exp_wt; Weight the sample will be expanded to (default is the landing weight). Pounds.

fr_bothn 3 Integer.
[O] fr_bothn; Total number of frames in postprocessing sample.

fr_fmnum 3 Integer.
[O] fr_fmnum; Total number of female frames in postprocessing sample.

fr_fmwt 8 Number (precision 7 and scale 1, example 123456.7).
[O] fr_fmwt; Total weight of female frames in postprocessing sample. Pounds.

fr_mnum 3 Integer.
[O] fr_mnum; Total number of male frames in postprocessing sample.

fr_mwt 8 Number (precision 7 and scale 1, example 123456.7).
[O] fr_mwt; Total weight of male frames in postprocessing sample. Pounds.

hours 5 Number (precision 4 and scale 1, example 123.4).
[O] hours; Hours fishing.

max_sizein 5 Number (precision 4 and scale 1, example 123.4).
[O] max_sizein; Maximum grade size (inches) for length graded species.

max_sizelb 5 Number (precision 4 and scale 1, example 123.4).
[O] max_sizelb; Maximum grade size (pounds) for weight graded species.

min_sizein 5 Number (precision 4 and scale 1, example 123.4).

[O] min_sizein; Minimum grade size (inches) for length graded species.

min_sizelb 5 Number (precision 4 and scale 1, example 123.4).
[O] min_sizelb; Minimum grade size (pounds) for weight graded species.

msec1 8 Number (precision 7 and scale 1, example 123456.7).
[O] msec1; First loran coordinate of catch.

msec2 8 Number (precision 7 and scale 1, example 123456.7).
[O] msec2; Second loran coordinate of catch.

nspec 3 Integer.
[O] nspec; Number of specimens of sample.

sample_quality 2 Alpha.
[O] sample_quality; applies to entire sample.
Values:
Sex
12 No sexes taken
Length
22 No lengths taken
Weight
32 Male or female total weights missing
33 Cluster weight missing
34 Sample weight missing
35 Category weight missing
36 Expansion weight missing
37 Expansion weight missing, estimated
38 Expansion weight differs from category weight
Miscellaneous
62 Disposition of catch missing
63 Fish Ticket unknown
64 Boatid unknown
65 Area of catch estimated
66 More than one PMFC area
67 Mixed gears
68 Size Grade estimated
69 Document No. unknown

71 Port unknown (70 is missing for now)
72 Sampler unknown

sampler 17 Text.
[O] sampler; sampler name(s).

start_bien 3 Integer.
[O] start_bien; Otolith tray number first otoliths go into.

state 2 Alpha.
[O] state; state area of catch (default is area of port if area of catch is not known).

tow 2 Alpha.
[O] tow; Trawl tow that was sampled (if known).

unk_num 3 Integer.
[O] unk_num; Total number of sex unknown fish in sample

unk_wt 8 Number (precision 7 and scale 1, example 123456.7).
[O] unk_wt; Total weight of sex unknown fish in sample in pounds.

vess_name 17 Text.
[O] vess_name; Vessel name.

Each SAMPLE transaction must be unique for: sample_no

The CLUSTER transaction consists of the following data items or attributes:

| DATA ITEM | SIZE | DESCRIPTION; RANGE OF VALUES |
|---|---|---|
| trans_id | 1 | Alpha. Required.<br>[WOCMF] Set to "H" => CLUSTER transaction. |
| sample_no | 14 | Alpha. Required.<br>Unique number for each sample.<br>[W] sample year + species id + sequence number.<br>[O] sample_no. |

[C] sample year + port code + quarter + sample number.
[M] sampyr + sampnum.
[F] year + portcode + sampletype + samplercode + samplenumber.

cluster_no 2 Integer. Required.
[W] sequential number for each cluster in sample.
[O] cluster.
[C] cluster number.
[M] clustnum.
[F] sequential number for each cluster in sample.

species_code 6 Alpha. Required.
Transferred to the coast-wide database as agency codes,
then converted to PacFIN species codes
[W] species id.
[O] sp_code.
[C] species code.
[M] "SABL" (constant).
[F] "PWHT" (constant).

cluster_wgt 9 Number (precision 8 and scale 2, example 123456.78).
[O] cluster_wt; weight of the cluster that the specimen
came from. Pounds.
[C] cluster weight; Total weight of fish in the
cluster. Pounds.
[M] clustwgt; Pounds.

cluster_wgt_um 1 Alpha. Required if cluster_wgt is present.
[OCM] Unit of measure (K = kg, L = lb)

species_wgt 9 Number (precision 8 and scale 2, example 123456.78).
[C] species weight; Total weight of fish of the given
species in the cluster. Pounds.

species_wgt_um 1 Alpha. Required if species_wgt is present.
[C] Unit of measure (K = kg, L = lb)

adj_clwt 7 Number (precision 6 and scale 1, example 12345.6).
[O] adj_clwt; Weight of the cluster that the specimen
came from minus weight of fish that can't be aged.

Pounds.

frame_clwt 7 Number (precision 6 and scale 1, example 12345.6).
[O] frame_clwt; Weight of the cluster that the specimen frame came from in postprocessing sample. Pounds.

Each CLUSTER transaction must be unique for: sample_no, cluster_no, species_code

The FISH transaction consists of the following data items or attributes:

| DATA ITEM | SIZE | DESCRIPTION; RANGE OF VALUES |
|---|---|---|
| trans_id | 1 | Alpha. Required.<br>[WOCMF] Set to "I" => FISH transaction. |
| sample_no | 14 | Alpha. Required.<br>Unique number for each sample.<br>[W] sample year + species id + sequence number.<br>[O] sample_no.<br>[C] sample year + port code + quarter + sample number.<br>[M] sampyr + sampnum.<br>[F] year + portcode + sampletype + samplercode + samplenumber. |
| cluster_no | 2 | Integer. Required.<br>[W] sequential number for each cluster in sample.<br>[O] cluster.<br>[C] cluster number.<br>[M] clustnum.<br>[F] sequential number for each cluster in sample. |
| species_code | 6 | Alpha. Required.<br>Transferred to the coast-wide database as agency codes, then converted to PacFIN species codes<br>[W] species id.<br>[O] sp_code.<br>[C] species code.<br>[M] "SABL" (constant).<br>[F] "PWHT" (constant). |

fish_no 5 Integer. Required.
[W] fish number.
[O] specimen; Number of specimen in sample.
[C] fish number.
[M] fishnum.
[F] specimennumber.

fish_age_final 3 Integer.
[W] best age.
[O] age; Age of specimen.
[C] sample year - year class.
[M] fage.
[F] age.

fish_length 5 Integer.
Lowest level data.
[W] length.
[O] length_cm or length_mm; Length of specimen in centimeters or millimeters.
[C] forklength; mm.
[M] len. mm.
[F] length. fork length in cm.

fish_length_type 1 Alpha. Required is fish_length is present.
Lowest level data.
[W] length type;
Values:
A = alternate length,
F = fork length,
S = standard length,
T = total length.
[O] "F" (constant).
[C] "F" (constant).
[M] Derived from cond.
Values:
D = Dorsal length (if cond = D),
F = Fork length (if cond = W).
[F] "F" (constant).

fish_length_um 1 Alpha. Required if fish_length or fork_length is

present.
[WOCMF] Unit of measure (C = cm, M = mm)

fish_weight 9 Number (precision 8 and scale 3, example 12345.678).
[W] weight; weight of fish in grams.
[O] weight_gm or weight_lb; Weight of specimen in grams or pounds.
[F] weight; weight in kg.

fish_weight_um 1 Alpha. Required if fish_weight is present.
[WOF] Unit of measure (G = gm, K = kg, L = lb)

fork_length 5 Integer.
[WOCMF] fork length of fish, either measured or estimated. Units in fish_length_um.

fork_length_estimated 1 Alpha.
[WOCMF] flag indicating source of fork_length;
Values:
T = True, fork_length was estimated,
F = False, fork_length was measured.

maturity_agcode 2 Alpha.
Will be converted to PacFIN codes (TBD).
[W] maturity; Values depend on species, contact WDFW for documentation of historical values, will be blank in future.
[O] maturity; Values depend on species group (rockfish, flatfish, etc.) but can be categorized as follows.
Values:
0 = unknown,
1-2 = immature,
3-8 = mature.
[C] maturity;
Values: female/male
1 = immature,
2 = early yolk/early maturity,
3 = late yolk/late maturity,
4 = eyed larvae/none,
5 = spent/none,

6 = recovery/none,
8 = undeterminable,
9 = not noted.
[F] maturity; not currently used.

sex_agcode 2 Alpha.
Will be converted to PacFIN codes (TBD).
[W] sex; 1 = male, 2 = female, 3 = unknown.
[O] sex; 1 = male, 2 = female, 9 = unknown.
[C] sex; 1 = male, 2 = female, 9 = unknown.
[M] sex; 1 = male, 2 = female, 9 = unsexed.
[F] sex; M = male, F = female, U = unknown.

brood_yr 4 Integer.
[O] brood_yr; Year of brooding (widow rockfish only).

fish_quality 2 Alpha.
[O] fish_quality; applies to individual fish samples.
Values:
11 = Individual sex missing, code sex as 9
21 = Individual length missing
31 = Individual weight missing, weights sampled
41 = Individual maturity missing, maturities sampled
51 = Individual age missing, sample aged

frame_wt 9 Number (precision 8 and scale 3, example 12345.678).
[O] frame_wt; Weight of specimen frame in postprocessing sample. Pounds.

freq 3 Integer.
[O] freq; Frequency (for non-aged fish).

Each FISH transaction must be unique for: sample_no, cluster_no, species_code, fish_no

The AGE transaction consists of the following data items or attributes:

| DATA ITEM | SIZE | DESCRIPTION; RANGE OF VALUES |
|---|---|---|

trans_id 1 Alpha. Required.
[WOCMF] Set to "J" => AGE transaction.

sample_no 14 Alpha. Required.
Unique number for each sample.
[W] sample year + species id + sequence number.
[O] sample_no.
[C] sample year + port code + quarter + sample number.
[M] sampyr + sampnum.
[F] year + portcode + sampletype + samplercode + samplenumber.

cluster_no 2 Integer. Required.
[W] sequential number for each cluster in sample.
[O] cluster.
[C] cluster number.
[M] clustnum.
[F] sequential number for each cluster in sample.

species_code 6 Alpha. Required.
Transferred to the coast-wide database as agency codes, then converted to PacFIN species codes
[W] species id.
[O] sp_code.
[C] species code.
[M] "SABL" (constant).
[F] "PWHT" (constant).

fish_no 5 Integer. Required.
[W] fish number.
[O] specimen; Number of specimen in sample.
[C] fish_no.
[M] fishnum.
[F] specimennumber.

age_no 2 Integer. Required.
[WOCMF] sequential number for each age.

age 3 Integer. Required.
[W] first age, second age, third age.

[O] age; Age of specimen.
[C] age; years.
[M] age 1-10.
[F] age; Age of fish in years.

age_method 1 Alpha.
[W] age method;
Values:
B = break and burn,
L = age derived from length,
N = not aged yet,
O = optical scanner,
S = surface read,
X = sectioning.

[O] age method;
Values:
1 = break and burn,
2 = surface read,
3 = scales,
4 = thin section,
5 = optical scanner,
6 = age derived from length,
9 = not read or unable to be aged.

age_readability 1 Alpha.
[W] age readability;
Values:
1 = poor,
2 = average,
3 = good

age_struct_agcode 1 Alpha.
Will be converted to PacFIN codes (TBD).
[W] age structure;
Values:
F = fin,
I = interoperculum,
N = no age structure taken,
O = otolith,

S = scale,
V = vertebra.
[O] struc_type;
Values:
1 = otoliths,
2 = interopercles,
3 = spines,
9 = lengths.
[C] "O" = otolith (constant)
[M] otclust1-4; whether otoliths were taken
Values: Y, N

aged_by 7 Text.
[W] aged by; Age reader

date_aged 8 Integer.
[W] date aged; for sample tracking; YYYYMMDD.

Each AGE transaction must be unique for: sample_no, cluster_no, species_code, fish_no, age_no

Modification log:
1. Added units and values for various data elements.
2. Transaction identifiers changed from S, C, F and A to G, H, I and J.
3. Added notes for elements that will be converted to PacFIN codes (TBD).
4. Changed name of pmfc_area to psmfc_area in SAMPLE transaction.
5. Added note that all unit of measure (um) elements will be converted to common units in the coast-wide database.
6. Moved age_method & age_structure from SAMPLE transaction to AGE transaction.
7. Removed logjoin from SAMPLE transaction.
8. Combined sort_group and grade elements in SAMPLE transaction.
9. Removed species_no from CLUSTER transaction.
10. Added species_code to list of unique elements for CLUSTER, FISH and AGE transactions.
11. Moved length_type from SAMPLE transaction to FISH transaction.
12. Added data type to description of WDFW sample_no.
13. Combined WDFW and CDFG data_type descriptions in SAMPLE transaction.
14. Added description of sample_strat in SAMPLE transaction.
15. Added paragraph to preamble describing update frequency and method.
16. Changed source codes in REPORT transaction to agency acronyms, and changed

source identifiers throughout the specification document to match PacFIN agency codes (agid).

17. Combined agency and data_source elements into sample_agency element in SAMPLE transaction.
18. Renamed inpfc to inpfc_area in SAMPLE transaction.
19. Added AGENCY-CODE-LIST transaction.
20. Renamed sample_type_wa to sample_type in SAMPLE transaction.
21. Combined sample_type_pw and sample_strat elements in SAMPLE transaction.
22. Added CDFG data elements cdfg_block, latitude and longitude to descriptions of block, latitude and longitude elements in SAMPLE transaction.
23. Combined dealer and plant_name elements in SAMPLE transaction.
24. Removed lat_deg, lat_min, lon_deg and lon_min from SAMPLE transaction.
25. Renamed condition to cond_agcode, disposition to disp_agcode, grade to grade_agcode, and fish_ticket_no to ftid in SAMPLE transaction.
26. Renamed length_type to fish_length_type, maturity to maturity_agcode and sex to sex_agcode in FISH transaction.
27. Renamed fish_age to age and age_structure to age_struct_agcode in AGE transaction.
28. Added WDFW sample condition element to description of cond_agcode in SAMPLE transaction.
29. Added WDFW port element to description of port in SAMPLE transaction.
30. Added WDFW sampler initials element to description of sampler_code in SAMPLE transaction.
31. Added fork_length and fork_length_estimated elements to FISH transaction.
32. Added paragraph to preamble describing AGENCY-CODE-LIST transactions.
33. Renamed boat_no to veid in SAMPLE transaction.
34. Increased size of sample_no and other_sample_no from 13 to 14 in all transactions.
35. Increased size of species_code from 5 to 6 in all transactions.
36. Moved species_wgt from SAMPLE transaction to CLUSTER transaction.
37. Added species_wgt_um to CLUSTER transaction.
38. Removed cluster_total_fish from CLUSTER transaction.
39. Moved quality from SAMPLE transaction to FISH transaction.
40. Renamed fr_fmnm to fr_fmnum and unk_wgt to unk_wt in SAMPLE transaction.
41. Removed keep from SAMPLE transaction (ODFW only).
42. Renamed quality to fish_quality in FISH transaction (ODFW only).
43. Added sample_quality to SAMPLE transaction (ODFW only).

# Appendix L: BDS Main Table Definitions

HEADER Table:

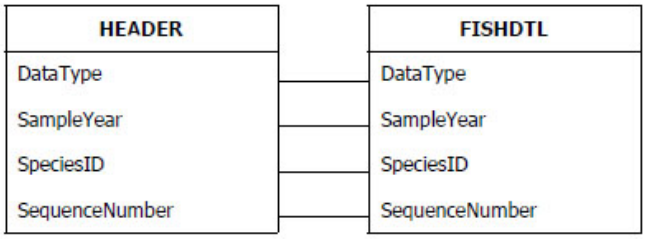
**Columns**

| Name | Type | Size |
|---|---|---|
| DataType | Text | 1 |
| SampleYear | Integer | 2 |
| SpeciesID | Text | 6 |
| SequenceNumber | Integer | 2 |
| FishUse | Text | 1 |
| AgeStructure | Text | 1 |
| AgeMethod1 | Text | 1 |
| AgeMethod2 | Text | 1 |
| AgeMethod3 | Text | 1 |
| EPUMultiplier | Double | 8 |
| SampleType | Text | 1 |
| SampleMethod | Text | 1 |
| SampleStrat | Text | 1 |
| SampleCondition | Text | 1 |
| LengthUnits | Text | 1 |
| LengthType | Text | 1 |
| WeightUnits | Text | 1 |
| VesselID | Text | 8 |
| FishTicket1 | Text | 8 |
| FishTicket2 | Text | 8 |
| GearType | Text | 2 |
| SportGearType | Text | 1 |
| SportGearMode | Text | 1 |
| MeshSize | Integer | 2 |
| DepthFished | Integer | 2 |
| Effort | Integer | 2 |
| Port | Text | 3 |
| Region | Text | 3 |
| PMFCArea | Text | 2 |
| MFSFArea | Integer | 2 |
| WDFArea | Integer | 2 |
| Ground | Integer | 2 |
| PunchCardArea | Integer | 2 |
| BaitfishArea | Integer | 2 |
| TowNumber | Integer | 2 |
| AcousticArea | Integer | 2 |
| SampleMonth | Integer | 2 |
| SampleDay | Integer | 2 |
| CatchWeight | Double | 8 |
| WeightSource | Text | 1 |
| SampleWeight | Integer | 2 |
| FishCount | Integer | 2 |
| SamplerInitials | Text | 8 |
| DateAged1 | Date/Time | 8 |
| AgedBy1 | Text | 7 |
| DateAged2 | Date/Time | 8 |
| AgedBy2 | Text | 7 |
| DateAged3 | Date/Time | 8 |
| AgedBy3 | Text | 7 |

| | | |
|---|---|---|
| WhereSent | Text | 4 |
| DateSentOut | Date/Time | 8 |
| DateReturn | Date/Time | 8 |
| LinkedSeqNo | Integer | 2 |
| PrimarySecondary | Text | 1 |
| DataSource | Text | 1 |
| OtherSampleNo | Text | 12 |
| Comments | Memo | - |
| Create_DFWuser_Id | Text | 12 |
| Create_DT | Date/Time | 8 |
| Modify_DFWuser_Id | Text | 12 |
| Modify_DT | Date/Time | 8 |
| Modify_TS | Binary | 8 |

**Relationships**

**HEADERFISHDTL**




| | |
|---|---|
| Attributes: | Not Enforced, Left Join |
| RelationshipType: | One-To-Many |

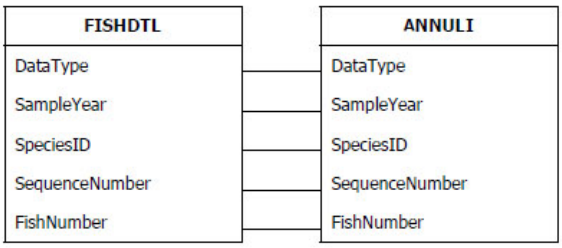
FISHDTL Table:

**Columns**

| Name | Type | Size |
|---|---|---|
| DataType | Text | 1 |
| SampleYear | Integer | 2 |
| SpeciesID | Text | 6 |
| SequenceNumber | Integer | 2 |
| FishNumber | Integer | 2 |
| Weight | Double | 8 |
| Length | Integer | 2 |
| Sex | Text | 1 |
| Maturity | Text | 1 |
| Extra | Text | 10 |
| FirstAge | Integer | 2 |
| SecondAge | Integer | 2 |
| ThirdAge | Integer | 2 |
| BestAge | Integer | 2 |
| AgeReadability | Text | 1 |
| Outlier | Text | 1 |
| Create_DFWuser_Id | Text | 12 |
| Create_DT | Date/Time | 8 |
| Modify_DFWuser_Id | Text | 12 |
| Modify_DT | Date/Time | 8 |
| Modify_TS | Binary | 8 |

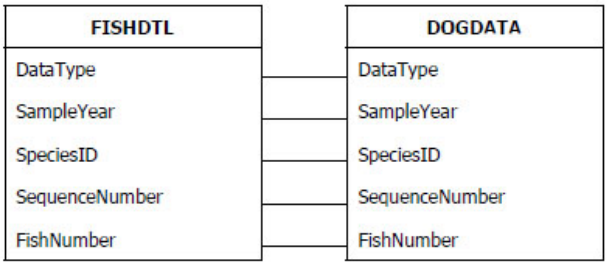
**Relationships**

**FISHDTLANNULI**



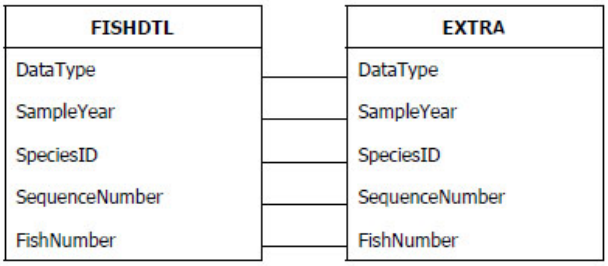

Attributes: Unique, Not Enforced, Left Join
RelationshipType: One-To-One

**FISHDTLDOGDATA**



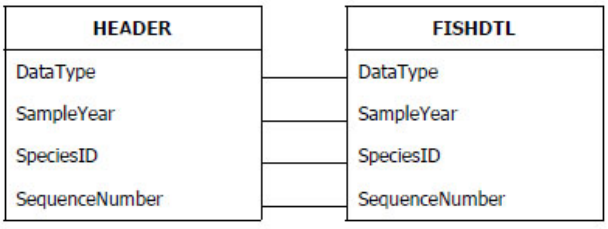

Attributes: Unique, Not Enforced, Left Join
RelationshipType: One-To-One

**FISHDTLEXTRA**




Attributes: Unique, Not Enforced, Left Join
RelationshipType: One-To-One

**HEADERFISHDTL**




Attributes: Not Enforced, Left Join
RelationshipType: One-To-Many

ANNULI Table:

**Columns**

| Name | Type | Size |
|---|---|---|
| DataType | Text | 1 |
| SampleYear | Integer | 2 |
| SpeciesID | Text | 6 |
| SequenceNumber | Integer | 2 |
| FishNumber | Integer | 2 |
| FirstAnnulus | Integer | 2 |
| SecondAnnulus | Integer | 2 |
| ThirdAnnulus | Integer | 2 |
| LastAnnulus | Integer | 2 |
| Edge | Integer | 2 |
| Comments | Text | 24 |
| Create_DFWuser_Id | Text | 12 |
| Create_DT | Date/Time | 8 |
| Modify_DFWuser_Id | Text | 12 |
| Modify_DT | Date/Time | 8 |
| Modify_TS | Binary | 8 |

DOGDATA Table:

**Columns**

| Name | Type | Size |
|---|---|---|
| DataType | Text | 1 |
| SampleYear | Integer | 2 |
| SpeciesID | Text | 6 |
| SequenceNumber | Integer | 2 |
| FishNumber | Integer | 2 |
| Measure1 | Double | 8 |
| Measure2 | Double | 8 |
| Measure3 | Double | 8 |
| RingCnt1 | Integer | 2 |
| RingCnt2 | Integer | 2 |
| MissingAge1 | Double | 8 |
| MissingAge2 | Double | 8 |
| Sex | Integer | 2 |
| Comments | Text | 24 |
| Create_DFWuser_id | Text | 12 |
| Create_DT | Date/Time | 8 |
| Modify_DFWuser_id | Text | 12 |
| Modify_DT | Date/Time | 8 |
| Modify_TS | Binary | 8 |

EXTRA Table:

**Columns**

| Name | Type | Size |
|---|---|---|
| DataType | Text | 1 |
| SampleYear | Integer | 2 |
| SpeciesID | Text | 6 |
| SequenceNumber | Integer | 2 |
| FishNumber | Integer | 2 |
| Barcode | Text | 17 |
| VialFishN | Decimal | 16 |
| FDAgeStructure | Text | 1 |
| PITTTagN | Text | 16 |

AUDIT Table:

**Columns**

| Name | Type | Size |
|---|---|---|
| TableID | Text | 1 |
| ActionType | Text | 1 |
| DataType | Text | 1 |
| SampleYear | Integer | 2 |
| SpeciesID | Text | 6 |
| SequenceNumber | Integer | 2 |
| FishNumber | Integer | 2 |
| UserName | Text | 12 |
| ActionDT | Date/Time | 8 |

# Appendix M: BDS Code List Look-Up Tables

AGEMETH:

| Code | Description |
|---|---|
| B | Break and burn |
| E | Enhanced break and burn |
| L | Age derived from length |
| M | More than one agemeth (as BB+SR) |
| N | Not aged yet |
| O | Optical scanner |
| S | Surface read (including scales) |
| U | Agemeth not recorded |
| X | Sectioning |

AGESTRUC:

| Code | Description |
|---|---|
| D | Second dorsal spine |
| F | Fin |
| I | Interoperculum |
| N | No age structure |
| O | Otolith |
| S | Scale |
| V | Vertebra |

DATSOURC:

| Code | Description |
|---|---|
| A | NMFS, Sand Point |
| B | NMFS, NWFSC |
| C | CDFG |
| D | Dept of Fish & |
| E | MRD emphasis |
| F | NMFS, WCGOP |
| G | Genetic study |
| I | Int. Pacific Halibut |
| M | MRFSS (RecFIN) |
| N | NMFS, Newport |
| O | ODFW |
| P | Puget Sound |
| S | WDFW salmon |
| T | NMFS, Tiburon |
| W | WDFW groundfish |
| Z | Makah tribal |

GEAR:

| Code | Description |
|---|---|
| HB | Hook and line, boat |
| HC | Hook and line, charter |
| HP | Hook and line, pier |
| L? | Unknown line (general) |
| LJ | Handline jig |
| LL | Long line (set line) |
| N? | Unknown misc. net |
| ND | Dipbag net |
| NG | Gillnet (setnet) |
| P? | Unknown pot (general) |
| PB | Bottomfish pot |
| PH | Hagfish pot |
| R? | Unknown troll (general) |
| RB | Bottomfish troll |
| RS | Salmon troll |
| S? | Unknown seine (general) |
| SB | Beach seine |
| SL | Lampara |
| SP | Purse seine |
| T? | Unknown trawl (general) |
| TB | Bottom trawl |
| TF | Selective flatfish trawl |
| TH | Trawl - hake |
| TL | Rope (line) trawl |
| TM | Midwater trawl |
| TP | Paired or danish trawl |
| TR | Roller trawl |
| TS | Shrimp trawl |
| TW | Wing trawl |
| W? | Unknown weir (general) |
| WF | Fish weir |

LENTYPE:

| Code | Description |
|---|---|
| A | Alternate |
| F | Fork length |
| R | Skate spiracle |
| S | Standard |
| T | Total length |

LENUNITS:

| Code | Description |
|---|---|
| C | Centimeters |
| M | Millimeters |

OUTLIER:

| Code | Description |
|---|---|
| 1 | Best age filled by algorithm, 1 age |
| 2 | Best age filled by algorithm, 2 ages |
| 3 | Best age filled by algorithm, 3 ages |
| 9 | Best age not filled, age difference > 1 |
| M | Dogfish missing age algorithm applied |

READABIL:

| Code | Description |
|---|---|
| 1 | Excellent, clear pattern |
| 2 | Average structure |
| 3 | Difficult structure |
| 4 | Minimum age only given-difficult to assign |
| 5 | Not aged-structure not discernable |
| 6 | Not aged-process storage or collectors |
| 7 | Double read structure |

REGION:

| Code | Description |
|---|---|
| CHR | Charlotte, Canada |
| COL | Columbia |
| CON | Conception |
| CS | Central Sound |
| EUR | Eureka |
| GB | Gulf-Bellingham |
| HC | Hood Canal |
| JF | Juan de Fuca |
| MON | Monterrey |
| PS | Puget Sound |
| SEA | Southeast Alaska |
| SJ | San Juans |
| SS | South Sound |
| VCN | Vancouver, Canada |
| VUS | Vancouver, U.S. |
| WJ | West Juan de Fuca |

SAMPMETH:

| Code | Description |
|---|---|
| N | Systematic |
| P | Purposive |
| R | Random |
| S | Stratified |
| X | Special |

SAMPSTRA:

| Code | Description |
|---|---|
| A | Age |
| L | Length |
| M | Maturity |
| W | Weight |

SAMPTYPE:

| Code | Description |
|---|---|
| C | Commercial |
| M | Market |
| R | Research |
| S | Special |

WTSOURCE:

| Code | Description |
|---|---|
| B | Buyer's weight (fish ticket) |
| C | Cruise data |
| F | Fisher's estimate |
| S | Sampler's estimate |

WTUNITS:

| Code | Description |
|---|---|
| G | Grams |
| H | Hundredths of pounds |
| P | Pounds |
| T | Tenths of ounces |

PORT:

| PortCode | PortName |
|---|---|
| ABE | Aberdeen |
| ALA | Alaska (all ports) |
| ANA | Anacortes |
| AST | Astoria |
| B.B | Bodega Bay |
| B.C | Bay Center |
| BAN | Bandon |
| BEL | Bellingham |
| BLA | Blaine |
| BRE | Bremerton |
| BRI | Brinnon |
| BRO | Brookings |
| C.B | Copalis Beach |
| CAL | California (all ports) |
| CHA | Charleston |
| CHI | Chinook |
| CNB | Canada, B |
| COO | Coos Bay |
| COU | Coupeville |
| CRS | Crescent City, CA |
| D.B | Depoe Bay |
| EUR | Eureka |
| EVE | Everett |
| F.B | Fort Bragg |
| F.H | Friday Harbor |
| FLO | Florence |
| G.B | Gold Beach |
| GAR | Garibaldi |
| HAM | Hammond |
| HOQ | Hoquiam |
| ILW | Ilwaco |
| KEL | Kelso |
| L.B | Long Beach |
| L.C | La Conner |
| L.P | La Push |
| LAC | La Conner |
| M.L | Moss Landing, CA |
| MNT | Monterey, CA |
| N.B | Neah Bay |
| NAS | Naselle |
| NEW | Newport |
| O.S | Ocean Shores |
| OLY | Olympia |
| ORE | Oregon (all ports) |
| P.A | Port Angeles |
| P.C | Pacific City |
| P.O | Port Orford |
| P.R | Point Roberts |
| P.T | Port Townsend |
| POU | Poulsbo |
| RAY | Raymond |
| S.B | South Bend |
| SEA | Seattle |
| SEK | Sekiu |
| SEQ | Sequim |
| SHE | Shelton |
| TAC | Tacoma |
| TAH | Taholah |
| TOK | Tokeland |
| VAN | Vancouver (Wash.) |
| WAR | Warrenton |
| WES | Westport |

VESSEL:

| FederalID | Name |
|---|---|
| 207 331 | ROVER |
| 207 980 | STING RAY |
| 208 350 | LITUYAH |
| 208 755 | AALESUND |
| 208755 | FISH HOG |
| 209 487 | TORDENSKJOLD |
| 209 491 | NERIED |
| 209 613 | OREGONIAN |
| 209 708 | ELEONORA |
| 209 724 | ELK |
| 209 773 | MARIE ANN GAIL |
| 209 855 | HYDRA |
| 210 212 | CONFIDENCE |
| 210 939 | SEYMOUR |
| 211 243 | GLORY |
| 211 400 | MERMAID II |
| 211 500 | ATICA |
| 211 813 | KODIAK |
| 212 376 | CONGRESS |
| 212 882 | ARIZONA |
| 212 947 | JOSEPH |
| 212 958 | NEW YORK |
| 213 088 | PROVIDENCE |
| 213 096 | STANLEY |
| 213 101 | SOLTA |
| 213 184 | AMAK |
| 213 193 | CHARLENE B |
| 213 309 | PT DEFIANCE |
| 213 386 | OCEAN QUEEN |
| 214 283 | SERENITY |
| 214 732 | CAPE SPENCER |
| 214 795 | PANTHER |

*Partial vessel list.

SPECIES:

| SpeciesID | CommonName | ScientificName |
|---|---|---|
| 090000 | Hagfish | (general) |
| 090101 | Pacific hagfish | Eptatretus stoutii |
| 090102 | Black hagfish | Eptatretus deani |
| 100000 | Cartilaginous fish | (general) |
| 110000 | Shark | (general) |
| 110101 | Sixgill shark | Hexanchus griseus |
| 110201 | Threasher shark | Alopias vulpinus |
| 110301 | Brown cat shark | Apristurus brunneus |
| 110401 | Soupfin shark | Galeorhinus zyopterus |
| 110501 | Blue shark | Prionace glauca |
| 110601 | Spiny dogfish | Squalus acanthias |
| 120000 | Skate or ray | (general) |
| 120101 | Big skate | Raja binoculata |
| 120102 | Longnose skate | Raja rhina |
| 130000 | Chaemera | (general) |
| 130101 | Ratfish | Hydrolagus colliei |
| 200000 | Primitive bony fish | (general) |
| 210000 | Sturgeon | (general) |
| 210101 | Green sturgeon | Acipenser medirostris |
| 210102 | White sturgeon | Acipenser transmontanus |
| 220000 | Baitfish | (general) |
| 220101 | American shad | Alosa sapidissima |
| 220201 | Pacific herring | Clupea harengus pallasi |
| 220301 | Pacific sardine | Sardinops sagax |
| 220401 | Northern anchovy | Engraulis mordax |
| 220501 | Surf smelt | Hypomesus pretiosus |
| 220601 | Eulachon | Thaleichthys pacificus |
| 230100 | Salmonid | (general) |
| 230200 | Trout or char | (general) |
| 300000 | Soft-bodied fish | (general) |
| 310000 | Toadfish | (general) |
| 310101 | Plainfin midshipman | Porichthys notatus |
| 320000 | Codfish | (general) |
| 320101 | Pacific cod | Gadus macrocephalus |
| 320201 | Pacific tomcod | Microgadus proximus |
| 320301 | Walleye pollock | Theragra chalcogramma |
| 320401 | Pacific whiting (hake) | Merluccius productus |
| 330000 | Rattail | (general) |
| 400000 | Perch-like fish | (general) |
| 410000 | Rockfish | (general) |
| 410101 | Rougheye rockfish | Sebastes aleutianus |
| 410102 | Pacific Ocean perch | Sebastes alutus |
| 410103 | Brown rockfish | Sebastes auriculatus |
| 410104 | Aurora rockfish | Sebastes aurora |
| 410105 | Redbanded rockfish | Sebastes babcocki |

| | | |
|---|---|---|
| 440401 | Great sculpin | Myoxocephalus |
| 440501 | Cabezon | Scorpaenichthys |
| 450000 | Jack or drum | (general) |
| 450101 | Jack mackerel | Trachurus symmetricus |
| 450201 | White seabass | Cynoscion nobilis |
| 460000 | Surfperch | (general) |
| 460101 | Shiner surfperch | Cymatogaster aggregata |
| 460201 | Striped seaperch | Embiotoca lateralis |
| 460301 | Silver seaperch | Hyperprosopon |
| 460401 | White seaperch | Phanerodon furcatus |
| 460501 | Pile perch | Rhacochilus vacca |
| 460601 | Redtail surfperch | Amphistichus rhodotersu |
| 470000 | Misc. perch-like fish | (general) |
| 470101 | Wolf-eel | Anarrichthys ocellatus |
| 470201 | Pacific sand lance | Ammodytes hexapterus |
| 470301 | Snake prickleback | Lumpenus sagitta |
| 480000 | Tuna or mackerel | (general) |
| 480101 | Chub mackerel | Scomber japonicus |
| 480201 | Albacore | Thunnus alalunga |
| 490000 | Flatfish | (general) |
| 490101 | Pacific sanddab | Citharichthys sordidus |
| 490102 | Speckled sanddab | Citharichthys stigmaeus |
| 490201 | Arrowtooth flounder | Atheresthes stomias |
| 490301 | Petrale sole | Eopsetta jordani |
| 490401 | Rex sole | Glyptocephalus zachirus |
| 490501 | Flathead sole | Hippoglossoides |
| 490601 | Butter sole | Isopsetta isolepis |
| 490701 | Rock sole | Lepidopsetta bilineata |
| 490801 | Dover sole | Microstomus pacificus |
| 490901 | English sole | Parophrys vetulus |
| 491001 | Starry flounder | Platichthys stellatus |
| 491101 | C-O sole | Pleuronichthys coenosus |
| 491201 | Sand sole | Psettichthys |
| 491301 | Pacific halibut | Hippoglossus stenolepis |
| 491401 | Slender sole | Lyopsetta exilis |
| 510000 | Unidentifiable food | (in the round) |
| 520000 | Unidentifiable food | (fillets) |
| 600000 | Shellfish | (general) |
| 610000 | Scallop | |
| 620000 | Squid | |
| 630000 | Octopus | |
| 710000 | Sea cucumber | |
| 420201 | Lingcod | Ophiodon elongatus |
| 430101 | Sablefish | Anaplopoma fimbria |
| 440000 | Sculpin | (general) |
| 440101 | Buffalo sculpin | Enophrys bison |
| 440201 | Red Irish lord | Hemilepidotus |
| 440202 | Brown Irish lord | Hemilepidotus spinosus |
| 440301 | Pacific staghorn sculpin | Leptocottus armatus |

# Appendix N: VBA Code Embedded in BDS User Interface

```
******************************
***VBA code in frmCommerc.
******************************
Public intSampYr As Integer, intSeqNo As Integer
Public strSpecID As String

Public Function CheckAgeCodes(AgeMeth As String) As Boolean
   Select Case Age_Structure
      Case "O"
         If AgeMeth = "B" Or AgeMeth = "E" Or AgeMeth = "S" Or AgeMeth = "N" Then
            CheckAgeCodes = True
         Else
            CheckAgeCodes = False
         End If
      Case "S"
         If AgeMeth = "L" Or AgeMeth = "S" Or AgeMeth = "N" Then
            CheckAgeCodes = True
         Else
            CheckAgeCodes = False
         End If
      Case "F"
         If AgeMeth = "X" Or AgeMeth = "N" Then
            CheckAgeCodes = True
         Else
            CheckAgeCodes = False
         End If
      Case "I", "V", "D"
         If AgeMeth = "S" Or AgeMeth = "N" Then
            CheckAgeCodes = True
         Else
            CheckAgeCodes = False
         End If
      Case "N"
         AgeMeth = Null
         CheckAgeCodes = True
   End Select
End Function
```

```
Public Sub FixAreaCodes()
   If Not IsNull(WDF_Area) Or Not IsNull(MF_SF_Area) Then
      If IsNull(WDF_Area) Then
         If Not FixCoastMFSF() Then
            MsgBox ("Invalid MF/SF area code.")
            MF_SF_Area = Null
            Ground = Null
         End If
      Else
         Select Case WDF_Area
            Case 20 To 29
               If Not IsNull(Ground) Then
                  If Not FixPuget() Then
                     MsgBox ("Catch area codes are incomplete or incompatible.")
                     Ground = Null
                  End If
               End If
            Case 83
               Ground = Null
               MF_SF_Area = WDF_Area
               PMFC_Area = "4A"
               Region = "PS "
               If IsNull(Sample_Year) Then
                  DoCmd.GoToControl "Sample Year"
               Else
                  DoCmd.GoToControl "Sample Month"
               End If
            Case Else
               If FixCoast() Then
                  Ground = Null
                  If IsNull(Sample_Year) Then
                     DoCmd.GoToControl "Sample Year"
                  Else
                     DoCmd.GoToControl "Sample Month"
                  End If
               Else
                  MsgBox ("Invalid WDF area code.")
                  WDF_Area = Null
                  Ground = Null
               End If
```

```
        End Select
      End If
    End If
End Sub

Public Function FixCoast() As Boolean
    Select Case WDF_Area
      Case 1
        MF_SF_Area = 56
        PMFC_Area = "5C"
        Region = "CHR"
        FixCoast = True
      Case 2
        MF_SF_Area = 56
        PMFC_Area = "5B"
        Region = "CHR"
        FixCoast = True
      Case 3
        MF_SF_Area = 56
        PMFC_Area = "5A"
        Region = "CHR"
        FixCoast = True
      Case 4 To 6
        MF_SF_Area = 57
        PMFC_Area = "3D"
        Region = "VCN"
        FixCoast = True
      Case 7, 8
        MF_SF_Area = 57
        PMFC_Area = "3C"
        Region = "VCN"
        FixCoast = True
      Case 9 To 11
        MF_SF_Area = 58
        PMFC_Area = "3C"
        Region = "VCN"
        FixCoast = True
      Case 12
        MF_SF_Area = 59
        PMFC_Area = "3B"
```

Region = "VUS"
FixCoast = True

Case 13
MF_SF_Area = 58
PMFC_Area = "3C"
Region = "VUS"
FixCoast = True

Case 14, 15
MF_SF_Area = 59
PMFC_Area = "3B"
Region = "VUS"
FixCoast = True

Case 16, 17
MF_SF_Area = 60
PMFC_Area = "3A"
Region = "COL"
FixCoast = True

Case 30
MF_SF_Area = 61
PMFC_Area = "3A"
Region = "COL"
FixCoast = True

Case 31
MF_SF_Area = 56
PMFC_Area = "5E"
Region = "CHR"
FixCoast = True

Case 32
MF_SF_Area = 61
PMFC_Area = "2C"
Region = "COL"
FixCoast = True

Case 33
MF_SF_Area = 55
PMFC_Area = "6A"
Region = "SEA"
FixCoast = True

Case 34
MF_SF_Area = 61
PMFC_Area = "2B"

```
            Region = "COL"
            FixCoast = True
        Case 36
            MF_SF_Area = 61
            PMFC_Area = "2A"
            Region = "EUR"
            FixCoast = True
        Case 38
            MF_SF_Area = 62
            PMFC_Area = "1C"
            Region = "EUR"
            FixCoast = True
        Case 40
            PMFC_Area = "1B"
            Region = "MON"
            FixCoast = True
        Case 42
            MF_SF_Area = 63
            PMFC_Area = "1A"
            Region = "CON"
            FixCoast = True
        Case 99
            MF_SF_Area = 56
            PMFC_Area = "5D"
            Region = "CHR"
            FixCoast = True
        Case Else
            FixCoast = False
    End Select
End Function

Public Function FixCoastMFSF() As Boolean
    Select Case MF_SF_Area
        Case 55
            PMFC_Area = "6A"
            Region = "SEA"
            FixCoastMFSF = True
        Case 56
            PMFC_Area = Null
            Region = "CHR"
```

```
            FixCoastMFSF = True
        Case 57
            PMFC_Area = Null
            Region = "VCN"
            FixCoastMFSF = True
        Case 58
            PMFC_Area = "3C"
            Region = Null
            FixCoastMFSF = True
        Case 59
            PMFC_Area = "3B"
            Region = "VUS"
            FixCoastMFSF = True
        Case 60
            PMFC_Area = "3A"
            Region = "COL"
            FixCoastMFSF = True
        Case 61 To 63
            PMFC_Area = Null
            Region = Null
            FixCoastMFSF = True
        Case Else
            FixCoastMFSF = False
    End Select
End Function

Public Function FixPuget() As Boolean
    Select Case WDF_Area
        Case 20
            Select Case Ground
                Case 10 To 19
                    MF_SF_Area = WDF_Area
                    PMFC_Area = "4A"
                    Region = "GB "
                    FixPuget = True
                Case 20 To 29
                    MF_SF_Area = WDF_Area
                    PMFC_Area = "4A"
                    Region = "SJ "
                    FixPuget = True
```

```
        Case Else
            If IsNull(Ground) Then
                MF_SF_Area = WDF_Area
                PMFC_Area = "4A"
                Region = Null
                FixPuget = True
            Else
                FixPuget = False
            End If
    End Select
Case 21
    If Ground > 9 And Ground < 30 Then
        MF_SF_Area = WDF_Area
        PMFC_Area = "4A"
        Region = "GB "
        FixPuget = True
    Else
        FixPuget = False
    End If
Case 22
    If Ground > 9 And Ground < 30 Then
        MF_SF_Area = WDF_Area
        PMFC_Area = "4A"
        Region = "SJ "
        FixPuget = True
    Else
        FixPuget = False
    End If
Case 23
    If Ground > 9 And Ground < 50 Then
        MF_SF_Area = WDF_Area
        PMFC_Area = "4A"
        Region = "JF "
        FixPuget = True
    Else
        FixPuget = False
    End If
Case 24
    If Ground > 9 And Ground < 60 Then
        MF_SF_Area = WDF_Area
```

```
        PMFC_Area = "4A"
        Region = "CS "
        FixPuget = True
    Else
        FixPuget = False
    End If
Case 25
    Select Case Ground
        Case 10 To 19
            MF_SF_Area = WDF_Area
            PMFC_Area = "4A"
            Region = "JF "
            FixPuget = True
        Case 20 To 29
            MF_SF_Area = WDF_Area
            PMFC_Area = "4A"
            Region = "CS "
            FixPuget = True
        Case 30 To 39
            MF_SF_Area = WDF_Area
            PMFC_Area = "4A"
            Region = "HC "
            FixPuget = True
        Case 40 To 49
            MF_SF_Area = WDF_Area
            PMFC_Area = "4A"
            Region = "CS "
            FixPuget = True
        Case 50 To 59
            MF_SF_Area = WDF_Area
            PMFC_Area = "4A"
            Region = "JF "
            FixPuget = True
        Case Else
            If IsNull(Ground) Then
                MF_SF_Area = WDF_Area
                PMFC_Area = "4A"
                Region = Null
                FixPuget = True
            Else
```

```
            FixPuget = False
        End If
    End Select
Case 26
    Select Case Ground
        Case 10 To 39
            MF_SF_Area = WDF_Area
            PMFC_Area = "4A"
            Region = "CS "
            FixPuget = True
        Case 40 To 49
            MF_SF_Area = WDF_Area
            PMFC_Area = "4A"
            Region = "SS "
            FixPuget = True
        Case Else
            FixPuget = False
    End Select
Case 27
    If Ground > 9 And Ground < 40 Then
        MF_SF_Area = WDF_Area
        PMFC_Area = "4A"
        Region = "HC "
        FixPuget = True
    Else
        FixPuget = False
    End If
Case 28
    If Ground > 9 And Ground < 50 Then
        MF_SF_Area = WDF_Area
        PMFC_Area = "4A"
        Region = "SS "
        FixPuget = True
    Else
        FixPuget = False
    End If
Case 29
    If Ground < 10 Then
        MF_SF_Area = WDF_Area
        PMFC_Area = "4A"
```

```
            Region = "WJ "
            FixPuget = True
        Else
            FixPuget = False
        End If
    Case Else
        FixPuget = False
  End Select
End Function
Public Sub CopyHeader()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset
  Set db = CurrentDb
  Set rst = db.OpenRecordset("HEADERX")
  With rst
    .Edit
    !DataType = Data_Type.Value
    !SampleYear = Sample_Year.Value
    !SpeciesID = Species_ID.Value
    !SequenceNumber = Sequence_Number.Value
    !FishUse = Fish_Use.Value
    !AgeStructure = Age_Structure.Value
    !AgeMethod1 = Age_Method1.Value
    !AgeMethod2 = Age_Method2.Value
    !AgeMethod3 = Age_Method3.Value
    !EPUMultiplier = EPU_Multiplier.Value
    !SampleType = Sample_Type.Value
    !SampleMethod = Sample_Method.Value
    !SampleStrat = Sample_Strat.Value
    !LengthUnits = Length_Units.Value
    !LengthType = Length_Type.Value
    !WeightUnits = Weight_Units.Value
    !VesselID = Vessel_ID.Value
    !FishTicket1 = Fish_Ticket_1.Value
    !FishTicket2 = Fish_Ticket_2.Value
    !GearType = Gear_Type.Value
    !SportGearType = Null
    !SportGearMode = Null
    !MeshSize = Mesh_Size.Value
    !DepthFished = Depth_Fished.Value
```

```
        !Port = Port.Value
        !Region = Region.Value
        !PMFCArea = PMFC_Area.Value
        !MFSFArea = MF_SF_Area.Value
        !WDFArea = WDF_Area.Value
        !Ground = Ground.Value
        !PunchCardArea = Null
        !TowNumber = Null
        !AcousticArea = Null
        !SampleMonth = Sample_Month.Value
        !SampleDay = Sample_Day.Value
        !CatchWeight = Catch_Weight.Value
        !WeightSource = Weight_Source.Value
        !SampleWeight = Null
        !FishCount = Fish_Count.Value
        !SamplerInitials = Sampler_Initials.Value
        !DateAged1 = Date_Aged1.Value
        !AgedBy1 = Aged_By1.Value
        !DateAged2 = Date_Aged2.Value
        !AgedBy2 = Aged_By2.Value
        !DateAged3 = Date_Aged3.Value
        !AgedBy3 = Aged_By3.Value
        !WhereSent = Where_Sent.Value
        !DateSentOut = Date_Sent_Out.Value
        !DateReturn = Date_Return.Value
        !DataSource = Data_Source.Value
        !OtherSampleNo = Other_Sample_No.Value
        !Comments = Comments.Value
        .Update
        .Close
    End With
End Sub

Private Sub Form_Open(Cancel As Integer)
    Me.FilterOn = True
End Sub

Private Sub Age_Method1_Enter()
    If Age_Structure = "N" Then
        DoCmd.GoToControl "EPU Multiplier"
```

```
    End If
End Sub

Private Sub Age_Method2_Enter()
    If Age_Structure = "N" Then
        DoCmd.GoToControl "EPU Multiplier"
    End If
End Sub

Private Sub Age_Method3_Enter()
    If Age_Structure = "N" Then
        DoCmd.GoToControl "EPU Multiplier"
    End If
End Sub

Private Sub Age_Method1_Exit(Cancel As Integer)
    If Not IsNull(Age_Structure) And Not IsNull(Age_Method1) Then
        If Not CheckAgeCodes(Age_Method1) Then
            MsgBox ("Age structure and method are incompatible.")
        End If
    End If
End Sub

Private Sub Age_Method2_Exit(Cancel As Integer)
    If Not IsNull(Age_Structure) And Not IsNull(Age_Method2) Then
        If Not CheckAgeCodes(Age_Method2) Then
            MsgBox ("Age structure and method are incompatible.")
        End If
    End If
End Sub

Private Sub Age_Method3_Exit(Cancel As Integer)
    If Not IsNull(Age_Structure) And Not IsNull(Age_Method3) Then
        If Not CheckAgeCodes(Age_Method3) Then
            MsgBox ("Age structure and method are incompatible.")
        End If
    End If
End Sub

Private Sub Age_Structure_Exit(Cancel As Integer)
```

```
    If Not IsNull(Age_Structure) And Not IsNull(Age_Method1) Then
        If Not CheckAgeCodes(Age_Method1) Then
            MsgBox ("Age structure and method are incompatible.")
        End If
    End If
End Sub

Private Sub Sample_Strat_Enter()
    If Sample_Method <> "S" Then
        DoCmd.GoToControl "Length Units"
    End If
End Sub

Private Sub WDF_Area_Exit(Cancel As Integer)
    Call FixAreaCodes
End Sub

Private Sub Ground_Exit(Cancel As Integer)
    Call FixAreaCodes
End Sub

Private Sub Locate_Sample_Button_Click()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim intCurrSampYr, intCurrSeqNo As Integer
    Dim strCurrSpecID As StringFormatEnum
    DoCmd.OpenForm "frmLocSamp", , , , , acDialog
    Set db = CurrentDb()
    Set rst = db.OpenRecordset("LOCSAMP", dbOpenSnapshot)
    If rst!DoLocate Then
        intCurrSampYr = Sample_Year.Value
        strCurrSpecID = Species_ID.Value
        intCurrSeqNo = Sequence_Number.Value
        intSampYr = rst!SampleYear
        strSpecID = rst!SpeciesID
        intSeqNo = rst!SequenceNumber
        Sample_Year.Enabled = True
        Sample_Year.SetFocus
        DoCmd.FindRecord intSampYr
        Species_ID.Enabled = True
```

```
      Species_ID.SetFocus
      If Species_ID.Value <> strSpecID Then
         DoCmd.FindRecord strSpecID, , , , , , False
      End If
      Sequence_Number.Enabled = True
      Sequence_Number.SetFocus
      If Sequence_Number.Value <> intSeqNo Then
         DoCmd.FindRecord intSeqNo, , , , , , False
      End If
      If Sample_Year.Value <> intSampYr Or Species_ID.Value <> _
      strSpecID Or Sequence_Number.Value <> intSeqNo Then
         MsgBox ("Sample doesn't exist!")
         Sample_Year.SetFocus
         DoCmd.FindRecord intCurrSampYr
         Species_ID.SetFocus
         If Species_ID.Value <> strCurrSpecID Then
            DoCmd.FindRecord strCurrSpecID, , , , , , False
         End If
         Sequence_Number.SetFocus
         If Sequence_Number.Value <> intCurrSeqNo Then
            DoCmd.FindRecord intCurrSeqNo, , , , , , False
         End If
      End If
      Sample_Year.Enabled = False
      Species_ID.Enabled = False
      Vessel_ID.SetFocus
      Sequence_Number.Enabled = False
   End If
   Vessel_ID.SetFocus
   rst.Close
End Sub

Private Sub Print_Sample_Button_Click()
   Call CopyHeader
   DoCmd.OpenReport "rptCommerc"
End Sub

Private Sub Save_and_Exit_Button_Click()
   DoCmd.Close
End Sub
```

```
Private Sub Copy_Header_Button_Click()
   Call CopyHeader
End Sub

Private Sub Paste_Header_Button_Click()
   Dim db As DAO.Database
   Dim rst As DAO.Recordset
   Dim Msg, Style, Title, Response
   Msg = "You are about to overwrite the data in the header. Do you want to continue?"
   Style = vbYesNo + vbExclamation + vbDefaultButton2
   Title = "Paste Header"
   Response = MsgBox(Msg, Style, Title)
   If Response = vbYes Then
      Set db = CurrentDb
      Set rst = db.OpenRecordset("HEADERX")
      With rst
         If !DataType = "C" Then
            Fish_Use.Value = !FishUse
            Age_Structure.Value = !AgeStructure
            Age_Method1.Value = !AgeMethod1
            Age_Method2.Value = !AgeMethod2
            Age_Method3.Value = !AgeMethod3
            EPU_Multiplier.Value = !EPUMultiplier
            Sample_Type.Value = !SampleType
            Sample_Method.Value = !SampleMethod
            Sample_Strat.Value = !SampleStrat
            Length_Units.Value = !LengthUnits
            Length_Type.Value = !LengthType
            Weight_Units.Value = !WeightUnits
            Vessel_ID.Value = !VesselID
            Fish_Ticket_1.Value = !FishTicket1
            Fish_Ticket_2.Value = !FishTicket2
            Gear_Type.Value = !GearType
            Mesh_Size.Value = !MeshSize
            Depth_Fished.Value = !DepthFished
            Port.Value = !Port
            Region.Value = !Region
            PMFC_Area.Value = !PMFCArea
            MF_SF_Area.Value = !MFSFArea
```

```
            WDF_Area.Value = !WDFArea
            Ground.Value = !Ground
            Sample_Month.Value = !SampleMonth
            Sample_Day.Value = !SampleDay
            Catch_Weight.Value = !CatchWeight
            Weight_Source.Value = !WeightSource
            Sampler_Initials.Value = !SamplerInitials
            Date_Aged1.Value = !DateAged1
            Aged_By1.Value = !AgedBy1
            Date_Aged2.Value = !DateAged2
            Aged_By2.Value = !AgedBy2
            Date_Aged3.Value = !DateAged3
            Aged_By3.Value = !AgedBy3
            Where_Sent.Value = !WhereSent
            Date_Sent_Out.Value = !DateSentOut
            Date_Return.Value = !DateReturn
            Data_Source.Value = !DataSource
            Other_Sample_No.Value = !OtherSampleNo
            Comments.Value = !Comments
            DoCmd.GoToControl "Vessel ID"
         Else
            Msg = "Last header data saved was not for this data type."
            Style = vbExclamation
            Title = "Paste Header"
            Response = MsgBox(Msg, Style, Title)
         End If
         .Close
      End With
   End If
End Sub

Private Sub Add_Annuli_Button_Click()
   Dim db As DAO.Database
   Dim rst As DAO.Recordset
   Dim intCurrSampYr, intCurrSeqNo, i As Integer
   Dim strCurrSpecID As StringFormatEnum
   Set db = CurrentDb
   Set rst = db.OpenRecordset("ANNULI")
   With rst
      For i = 1 To Fish_Count.Value
```

```
            .AddNew
            !DataType = Data_Type.Value
            !SampleYear = Sample_Year.Value
            !SpeciesID = Species_ID.Value
            !SequenceNumber = Sequence_Number.Value
            !FishNumber = i
            .Update
        Next
        .Close
    End With
    intCurrSampYr = Sample_Year.Value
    strCurrSpecID = Species_ID.Value
    intCurrSeqNo = Sequence_Number.Value
    Sample_Year.Enabled = True
    Sample_Year.SetFocus
    DoCmd.FindRecord intCurrSampYr
    Species_ID.Enabled = True
    Species_ID.SetFocus
    If Species_ID.Value <> strCurrSpecID Then
        DoCmd.FindRecord strCurrSpecID, , , , , , False
    End If
    Sequence_Number.Enabled = True
    Sequence_Number.SetFocus
    If Sequence_Number.Value <> intCurrSeqNo Then
        DoCmd.FindRecord intCurrSeqNo, , , , , , False
    End If
    Sample_Year.Enabled = False
    Species_ID.Enabled = False
    ANNULI.SetFocus
    Sequence_Number.Enabled = False
End Sub

******************************
***VBA code in frmFishDtl.
******************************
Public booDeleted As Boolean
Public intMaxFishNo As Integer

Sub GetMaxFish()
    Dim rst As DAO.Recordset
```

```
    Set rst = Me.RecordsetClone
    If rst.RecordCount > 0 Then
        rst.MoveLast
        intMaxFishNo = rst!FishNumber
    Else
        intMaxFishNo = 0
    End If
End Sub

Private Sub Form_AfterDelConfirm(Status As Integer)
    Call GetMaxFish
    Parent![Fish Count] = intMaxFishNo
    If Parent![Fish Count] > 0 Then
        DoCmd.RunCommand acCmdRecordsGoToPrevious
    End If
End Sub

Private Sub Form_Delete(Cancel As Integer)
    booDeleted = True
End Sub

Private Sub Form_Load()
    booDeleted = False
End Sub

Private Sub Weight_GotFocus()
    If IsNull([Fish Number]) Then
        If booDeleted = True Then
            booDeleted = False
        Else
            Call GetMaxFish
            Parent![Fish Count] = intMaxFishNo + 1
            [Fish Number] = Parent![Fish Count]
        End If
    End If
End Sub

Private Sub Length_GotFocus()
    If IsNull([Fish Number]) Then
        If booDeleted = True Then
```

```
            booDeleted = False
        Else
            Call GetMaxFish
            Parent![Fish Count] = intMaxFishNo + 1
            [Fish Number] = Parent![Fish Count]
        End If
    End If
End Sub

Private Sub Sex_GotFocus()
    If IsNull([Fish Number]) Then
        If booDeleted = True Then
            booDeleted = False
        Else
            Call GetMaxFish
            Parent![Fish Count] = intMaxFishNo + 1
            [Fish Number] = Parent![Fish Count]
        End If
    End If
End Sub

Private Sub Maturity_GotFocus()
    If IsNull([Fish Number]) Then
        If booDeleted = True Then
            booDeleted = False
        Else
            Call GetMaxFish
            Parent![Fish Count] = intMaxFishNo + 1
            [Fish Number] = Parent![Fish Count]
        End If
    End If
End Sub

Private Sub Extra_GotFocus()
    If IsNull([Fish Number]) Then
        If booDeleted = True Then
            booDeleted = False
        Else
            Call GetMaxFish
            Parent![Fish Count] = intMaxFishNo + 1
```

```
        [Fish Number] = Parent![Fish Count]
      End If
   End If
End Sub

Private Sub First_Age_GotFocus()
   If IsNull([Fish Number]) Then
      If booDeleted = True Then
         booDeleted = False
      Else
         Call GetMaxFish
         Parent![Fish Count] = intMaxFishNo + 1
         [Fish Number] = Parent![Fish Count]
      End If
   End If
End Sub

Private Sub Second_Age_GotFocus()
   If IsNull([Fish Number]) Then
      If booDeleted = True Then
         booDeleted = False
      Else
         Call GetMaxFish
         Parent![Fish Count] = intMaxFishNo + 1
         [Fish Number] = Parent![Fish Count]
      End If
   End If
End Sub

Private Sub Third_Age_GotFocus()
   If IsNull([Fish Number]) Then
      If booDeleted = True Then
         booDeleted = False
      Else
         Call GetMaxFish
         Parent![Fish Count] = intMaxFishNo + 1
         [Fish Number] = Parent![Fish Count]
      End If
   End If
End Sub
```

```
Private Sub Best_Age_GotFocus()
   If IsNull([Fish Number]) Then
      If booDeleted = True Then
         booDeleted = False
      Else
         Call GetMaxFish
         Parent![Fish Count] = intMaxFishNo + 1
         [Fish Number] = Parent![Fish Count]
      End If
   End If
End Sub

Private Sub Age_Readability_GotFocus()
   If IsNull([Fish Number]) Then
      If booDeleted = True Then
         booDeleted = False
      Else
         Call GetMaxFish
         Parent![Fish Count] = intMaxFishNo + 1
         [Fish Number] = Parent![Fish Count]
      End If
   End If
End Sub

Private Sub Weight_Label_DblClick(Cancel As Integer)
   If Weight_Label.ForeColor = -2147483630 Then
      Weight_Label.ForeColor = 8421504
      Weight.TabStop = False
   Else
      Weight_Label.ForeColor = -2147483630
      Weight.TabStop = True
   End If
End Sub

Private Sub Length_Label_DblClick(Cancel As Integer)
   If Length_Label.ForeColor = -2147483630 Then
      Length_Label.ForeColor = 8421504
      Length.TabStop = False
   Else
```

```
      Length_Label.ForeColor = -2147483630
      Length.TabStop = True
   End If
End Sub

Private Sub Sex_Label_DblClick(Cancel As Integer)
   If Sex_Label.ForeColor = -2147483630 Then
      Sex_Label.ForeColor = 8421504
      Sex.TabStop = False
   Else
      Sex_Label.ForeColor = -2147483630
      Sex.TabStop = True
   End If
End Sub

Private Sub Maturity_Label_DblClick(Cancel As Integer)
   If Maturity_Label.ForeColor = -2147483630 Then
      Maturity_Label.ForeColor = 8421504
      Maturity.TabStop = False
   Else
      Maturity_Label.ForeColor = -2147483630
      Maturity.TabStop = True
   End If
End Sub

Private Sub Extra_Label_DblClick(Cancel As Integer)
   If Extra_Label.ForeColor = -2147483630 Then
      Extra_Label.ForeColor = 8421504
      Extra.TabStop = False
   Else
      Extra_Label.ForeColor = -2147483630
      Extra.TabStop = True
   End If
End Sub

Private Sub First_Age_Label_DblClick(Cancel As Integer)
   If First_Age_Label.ForeColor = -2147483630 Then
      First_Age_Label.ForeColor = 8421504
      First_Age.TabStop = False
   Else
```

```
      First_Age_Label.ForeColor = -2147483630
      First_Age.TabStop = True
   End If
End Sub

Private Sub Second_Age_Label_DblClick(Cancel As Integer)
   If Second_Age_Label.ForeColor = -2147483630 Then
      Second_Age_Label.ForeColor = 8421504
      Second_Age.TabStop = False
   Else
      Second_Age_Label.ForeColor = -2147483630
      Second_Age.TabStop = True
   End If
End Sub

Private Sub Third_Age_Label_DblClick(Cancel As Integer)
   If Third_Age_Label.ForeColor = -2147483630 Then
      Third_Age_Label.ForeColor = 8421504
      Third_Age.TabStop = False
   Else
      Third_Age_Label.ForeColor = -2147483630
      Third_Age.TabStop = True
   End If
End Sub

Private Sub Best_Age_Label_DblClick(Cancel As Integer)
   If Best_Age_Label.ForeColor = -2147483630 Then
      Best_Age_Label.ForeColor = 8421504
      Best_Age.TabStop = False
   Else
      Best_Age_Label.ForeColor = -2147483630
      Best_Age.TabStop = True
   End If
End Sub

Private Sub Age_Readability_Label_DblClick(Cancel As Integer)
   If Age_Readability_Label.ForeColor = -2147483630 Then
      Age_Readability_Label.ForeColor = 8421504
      Age_Readability.TabStop = False
   Else
```

```
    Age_Readability_Label.ForeColor = -2147483630
    Age_Readability.TabStop = True
  End If
End Sub


******************************
***VBA code in frmStartUp.
******************************
Public strStartYr As String, strEndYr As String

Public Sub OpenBDSForm()
  If IsNull(Ending_Year_Input) Then
    strStartYr = Str(Starting_Year_Input)
    Select Case Data_Type_Option_Group
      Case 1
        DoCmd.OpenForm "frmCommerc", , , "SampleYear = " + strStartYr
      Case 2
        DoCmd.OpenForm "frmResearch", , , "SampleYear = " + strStartYr
      Case 3
        DoCmd.OpenForm "frmSport", , , "SampleYear = " + strStartYr
    End Select
  Else
    strStartYr = Str(Starting_Year_Input)
    strEndYr = Str(Ending_Year_Input)
    Select Case Data_Type_Option_Group
      Case 1
        DoCmd.OpenForm "frmCommerc", , , "SampleYear >= " + strStartYr + " and
SampleYear <= " + strEndYr
      Case 2
        DoCmd.OpenForm "frmResearch", , , "SampleYear >= " + strStartYr + " and
SampleYear <= " + strEndYr
      Case 3
        DoCmd.OpenForm "frmSport", , , "SampleYear >= " + strStartYr + " and SampleYear
<= " + strEndYr
    End Select
  End If
End Sub

Private Sub Add_Samples_Button_Click()
```

```
    DoCmd.OpenForm "frmAddSamp"
End Sub

Private Sub Edit_Data_Button_Click()
    If IsNull(Starting_Year_Input) Then
        MsgBox ("Must enter at least a starting year.")
    Else
        Call OpenBDSForm
    End If
End Sub

Private Sub Starting_Year_Input_Exit(Cancel As Integer)
    If Starting_Year_Input > Year(Date) Then
        MsgBox ("Starting year is later than current year, setting to current year.")
        Starting_Year_Input = Year(Date)
    End If
    If Not IsNull(Ending_Year_Input) Then
        If Starting_Year_Input > Ending_Year_Input Then
            MsgBox ("Starting year is later than ending year, setting to ending year.")
            Starting_Year_Input = Ending_Year_Input
        End If
        If (Ending_Year_Input - Starting_Year_Input) > 2 Then
            MsgBox ("Range of years is more than three, adjusting ending year.")
            Ending_Year_Input = Starting_Year_Input + 2
        End If
    End If
End Sub

Private Sub Ending_Year_Input_Exit(Cancel As Integer)
    If Int(Ending_Year_Input) > Year(Date) Then
        MsgBox ("Ending year is later than current year, setting to current year.")
        Ending_Year_Input = Year(Date)
    End If
    If Not IsNull(Starting_Year_Input) Then
        If Starting_Year_Input > Ending_Year_Input Then
            MsgBox ("Starting year is later than ending year, setting to ending year.")
            Starting_Year_Input = Ending_Year_Input
        End If
        If (Ending_Year_Input - Starting_Year_Input) > 2 Then
            MsgBox ("Range of years is more than three, adjusting ending year.")
```

```
        Ending_Year_Input = Starting_Year_Input + 2
      End If
   End If
End Sub

Private Sub Exit_Button_Click()
   DoCmd.Close
End Sub
```

**********

***VBA code in frmAddSamp.

**********

```
Public Sub AddSample()
   Dim db As DAO.Database
   Dim rst As DAO.Recordset
   Dim strDatTyp As String
   Dim strSampYr As String, strSpecID As String, strSQL As String
   Dim NextSeq As Integer
   Dim Msg, Style, Title, Response, MyString
   strDatTyp = Mid("CRS", Data_Type_Option_Group, 1)
   strSampYr = CStr(Sample_Year_Input)
   strSpecID = Species_ID_Input
   strSQL = "SELECT MAX(SequenceNumber) as MaxSeq INTO NEXTSEQ " & _
   "FROM HEADER WHERE DataType='" & strDatTyp & "' AND SampleYear=" & _
   strSampYr & " AND SpeciesID=" & "'" & strSpecID & "';"
   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   Set db = CurrentDb()
   Set rst = db.OpenRecordset("NEXTSEQ", dbOpenSnapshot)
   If IsNull(rst![MaxSeq]) Then
      Sequence_Number_Display = 1
   Else
      Sequence_Number_Display = rst![MaxSeq] + 1
   End If
   Msg = "Add this sample?"
   Style = vbYesNo + vbQuestion
   Title = "Add Sample"
   Response = MsgBox(Msg, Style, Title)
   If Response = vbYes Then
```

```
        strSQL = "INSERT INTO HEADER (DataType, SampleYear, " & _
        "SpeciesID, SequenceNumber) VALUES ('" & strDatTyp & _
        "', " & strSampYr & ", '" & strSpecID & "', " & _
        Sequence_Number_Display & ");"
        DoCmd.RunSQL strSQL
        Sample_Added_Display_1 = "Specimen label is: " & strSpecID & " " & _
        Right(strSampYr, 2) & Right("000" & Sequence_Number_Display, 3)
        Sample_Added_Display_2 = "Click 'Add' again to add another for this year/species."
    Else
        Sequence_Number_Display = Null
    End If
    rst.Close
    DoCmd.SetWarnings True
End Sub

Private Sub Add_Button_Click()
    Call AddSample
End Sub

Private Sub Close_Button_Click()
    DoCmd.Close
End Sub

Private Sub Data_Type_Option_Group_Enter()
    Sequence_Number_Display = Null
    Sample_Added_Display_1 = ""
    Sample_Added_Display_2 = ""
End Sub

Private Sub Sample_Year_Input_Enter()
    Sequence_Number_Display = Null
    Sample_Added_Display_1 = ""
    Sample_Added_Display_2 = ""
End Sub

Private Sub Species_ID_Input_Enter()
    Sequence_Number_Display = Null
    Sample_Added_Display_1 = ""
    Sample_Added_Display_2 = ""
End Sub
```

```
******************************
***VBA code in frmLocSamp.
******************************
Private Sub Cancel_Button_Click()
   [DoLocate] = False
   DoCmd.Close
End Sub

Private Sub OK_Button_Click()
   [DoLocate] = True
   DoCmd.Close
End Sub
```

# Appendix O: BDS Change Tracking Triggers and Update Procedure

```
**********************************************************************************************
***This code must be appended to existing code for cascade delete, cascade update, and
referential integrity.
**********************************************************************************************
ALTER TRIGGER dbo.HEADER_DTrig
ON dbo.HEADER FOR DELETE AS
/* * CHANGE TRACKING */
BEGIN
  INSERT INTO dbo.AUDIT
    SELECT 'H', 'D', DataType, SampleYear, SpeciesID, SequenceNumber, 0,
USER_NAME(USER_ID()), GETDATE() FROM deleted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE dbo.AUDIT.TableID = 'H'
AND dbo.AUDIT.ActionType = 'D' AND
    dbo.AUDIT.DataType = deleted.DataType AND dbo.AUDIT.SampleYear =
deleted.SampleYear AND
    dbo.AUDIT.SpeciesID = deleted.SpeciesID AND dbo.AUDIT.SequenceNumber =
deleted.SequenceNumber)
  UPDATE dbo.AUDIT SET dbo.AUDIT.UserName = USER_NAME(USER_ID()),
dbo.AUDIT.ActionDT = GETDATE() FROM deleted
    WHERE  dbo.AUDIT.TableID = 'H' AND dbo.AUDIT.ActionType = 'D' AND
dbo.AUDIT.DataType = deleted.DataType AND
    dbo.AUDIT.SampleYear = deleted.SampleYear AND dbo.AUDIT.SpeciesID =
deleted.SpeciesID AND
    dbo.AUDIT.SequenceNumber = deleted.SequenceNumber
END

ALTER TRIGGER dbo.HEADER_ITrig
ON dbo.HEADER FOR INSERT AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
BEGIN
  INSERT INTO dbo.AUDIT
    SELECT 'H', 'I', DataType, SampleYear, SpeciesID, SequenceNumber, 0,
USER_NAME(USER_ID()), GETDATE() FROM inserted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE dbo.AUDIT.TableID = 'H'
AND dbo.AUDIT.ActionType = 'I' AND
```

```
    dbo.AUDIT.DataType = inserted.DataType AND dbo.AUDIT.SampleYear =
inserted.SampleYear AND
    dbo.AUDIT.SpeciesID = inserted.SpeciesID AND dbo.AUDIT.SequenceNumber =
inserted.SequenceNumber)
  UPDATE dbo.AUDIT SET dbo.AUDIT.UserName = USER_NAME(USER_ID()),
dbo.AUDIT.ActionDT = GETDATE() FROM inserted
    WHERE  dbo.AUDIT.TableID = 'H' AND dbo.AUDIT.ActionType = 'I' AND
dbo.AUDIT.DataType = inserted.DataType AND
    dbo.AUDIT.SampleYear = inserted.SampleYear AND dbo.AUDIT.SpeciesID =
inserted.SpeciesID AND
    dbo.AUDIT.SequenceNumber = inserted.SequenceNumber
END

ALTER TRIGGER dbo.HEADER_UTrig
ON dbo.HEADER FOR UPDATE AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
IF (SELECT TrackUpdates FROM UTRIG) = 1
BEGIN
  INSERT INTO dbo.AUDIT
    SELECT 'H', 'U', DataType, SampleYear, SpeciesID, SequenceNumber, 0,
USER_NAME(USER_ID()), GETDATE() FROM inserted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE dbo.AUDIT.TableID = 'H'
AND dbo.AUDIT.ActionType = 'U' AND
    dbo.AUDIT.DataType = inserted.DataType AND dbo.AUDIT.SampleYear =
inserted.SampleYear AND
    dbo.AUDIT.SpeciesID = inserted.SpeciesID AND dbo.AUDIT.SequenceNumber =
inserted.SequenceNumber)
  UPDATE dbo.AUDIT SET dbo.AUDIT.UserName = USER_NAME(USER_ID()),
dbo.AUDIT.ActionDT = GETDATE() FROM inserted
    WHERE  dbo.AUDIT.TableID = 'H' AND dbo.AUDIT.ActionType = 'U' AND
dbo.AUDIT.DataType = inserted.DataType AND
    dbo.AUDIT.SampleYear = inserted.SampleYear AND dbo.AUDIT.SpeciesID =
inserted.SpeciesID AND
    dbo.AUDIT.SequenceNumber = inserted.SequenceNumber
END

ALTER TRIGGER dbo.FISHDTL_DTrig
ON dbo.FISHDTL FOR DELETE AS
SET NOCOUNT ON
```

```
/* * CHANGE TRACKING */
BEGIN
  INSERT INTO dbo.AUDIT
    SELECT 'F', 'D', DataType, SampleYear, SpeciesID, SequenceNumber, FishNumber,
USER_NAME(USER_ID()), GETDATE() FROM deleted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE dbo.AUDIT.TableID = 'F'
AND dbo.AUDIT.ActionType = 'D' AND
    dbo.AUDIT.DataType = deleted.DataType AND dbo.AUDIT.SampleYear =
deleted.SampleYear
    AND dbo.AUDIT.SpeciesID = deleted.SpeciesID AND dbo.AUDIT.SequenceNumber =
deleted.SequenceNumber
    AND dbo.AUDIT.FishNumber = deleted.FishNumber)
  UPDATE dbo.AUDIT SET UserName = USER_NAME(USER_ID()), ActionDT =
GETDATE() FROM deleted
    WHERE dbo.AUDIT.TableID = 'F' AND dbo.AUDIT.ActionType = 'D' AND
dbo.AUDIT.DataType = deleted.DataType AND
    dbo.AUDIT.SampleYear = deleted.SampleYear AND dbo.AUDIT.SpeciesID =
deleted.SpeciesID AND
    dbo.AUDIT.SequenceNumber = deleted.SequenceNumber AND dbo.AUDIT.FishNumber =
deleted.FishNumber
END

ALTER TRIGGER dbo.FISHDTL_ITrig
ON dbo.FISHDTL FOR INSERT AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
BEGIN
  INSERT INTO dbo.AUDIT
    SELECT 'F', 'I', DataType, SampleYear, SpeciesID, SequenceNumber, FishNumber,
USER_NAME(USER_ID()), GETDATE() FROM inserted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE dbo.AUDIT.TableID = 'F'
AND dbo.AUDIT.ActionType = 'I' AND
    dbo.AUDIT.DataType = inserted.DataType AND dbo.AUDIT.SampleYear =
inserted.SampleYear
    AND dbo.AUDIT.SpeciesID = inserted.SpeciesID AND dbo.AUDIT.SequenceNumber =
inserted.SequenceNumber
    AND dbo.AUDIT.FishNumber = inserted.FishNumber)
  UPDATE dbo.AUDIT SET UserName = USER_NAME(USER_ID()), ActionDT =
GETDATE() FROM inserted
```

```
    WHERE dbo.AUDIT.TableID = 'F' AND dbo.AUDIT.ActionType = 'I' AND
dbo.AUDIT.DataType = inserted.DataType AND
    dbo.AUDIT.SampleYear = inserted.SampleYear AND dbo.AUDIT.SpeciesID =
inserted.SpeciesID AND
    dbo.AUDIT.SequenceNumber = inserted.SequenceNumber AND dbo.AUDIT.FishNumber =
inserted.FishNumber
END

ALTER TRIGGER dbo.FISHDTL_UTrig
ON dbo.FISHDTL FOR UPDATE AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
IF (SELECT TrackUpdates FROM UTRIG) = 1
BEGIN
  INSERT INTO dbo.AUDIT
    SELECT 'F', 'U', DataType, SampleYear, SpeciesID, SequenceNumber, FishNumber,
USER_NAME(USER_ID()), GETDATE() FROM inserted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE dbo.AUDIT.TableID = 'F'
AND dbo.AUDIT.ActionType = 'U' AND
    dbo.AUDIT.DataType = inserted.DataType AND dbo.AUDIT.SampleYear =
inserted.SampleYear
    AND dbo.AUDIT.SpeciesID = inserted.SpeciesID AND dbo.AUDIT.SequenceNumber =
inserted.SequenceNumber
    AND dbo.AUDIT.FishNumber = inserted.FishNumber)
  UPDATE dbo.AUDIT SET UserName = USER_NAME(USER_ID()), ActionDT =
GETDATE() FROM inserted
    WHERE dbo.AUDIT.TableID = 'F' AND dbo.AUDIT.ActionType = 'U' AND
dbo.AUDIT.DataType = inserted.DataType AND
    dbo.AUDIT.SampleYear = inserted.SampleYear AND dbo.AUDIT.SpeciesID =
inserted.SpeciesID AND
    dbo.AUDIT.SequenceNumber = inserted.SequenceNumber AND dbo.AUDIT.FishNumber =
inserted.FishNumber
END

ALTER TRIGGER dbo.ANNULI_DTrig
ON dbo.ANNULI FOR DELETE AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
BEGIN
  INSERT INTO dbo.AUDIT
```

```
    SELECT 'A', 'D', DataType, SampleYear, SpeciesID, SequenceNumber, FishNumber,
USER_NAME(USER_ID()), GETDATE() FROM deleted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE dbo.AUDIT.TableID = 'A'
AND dbo.AUDIT.ActionType = 'D' AND
    dbo.AUDIT.DataType = deleted.DataType AND dbo.AUDIT.SampleYear =
deleted.SampleYear
    AND dbo.AUDIT.SpeciesID = deleted.SpeciesID AND dbo.AUDIT.SequenceNumber =
deleted.SequenceNumber
    AND dbo.AUDIT.FishNumber = deleted.FishNumber)
  UPDATE dbo.AUDIT SET UserName = USER_NAME(USER_ID()), ActionDT =
GETDATE() FROM deleted
    WHERE dbo.AUDIT.TableID = 'A' AND dbo.AUDIT.ActionType = 'D' AND
dbo.AUDIT.DataType = deleted.DataType AND
    dbo.AUDIT.SampleYear = deleted.SampleYear AND dbo.AUDIT.SpeciesID =
deleted.SpeciesID AND
    dbo.AUDIT.SequenceNumber = deleted.SequenceNumber AND dbo.AUDIT.FishNumber =
deleted.FishNumber
END

ALTER TRIGGER dbo.ANNULI_ITrig
ON dbo.ANNULI FOR INSERT AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
BEGIN
  INSERT INTO dbo.AUDIT
    SELECT 'A', 'I', DataType, SampleYear, SpeciesID, SequenceNumber, FishNumber,
USER_NAME(USER_ID()), GETDATE() FROM inserted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE dbo.AUDIT.TableID = 'A'
AND dbo.AUDIT.ActionType = 'I' AND
    dbo.AUDIT.DataType = inserted.DataType AND dbo.AUDIT.SampleYear =
inserted.SampleYear
    AND dbo.AUDIT.SpeciesID = inserted.SpeciesID AND dbo.AUDIT.SequenceNumber =
inserted.SequenceNumber
    AND dbo.AUDIT.FishNumber = inserted.FishNumber)
  UPDATE dbo.AUDIT SET UserName = USER_NAME(USER_ID()), ActionDT =
GETDATE() FROM inserted
    WHERE dbo.AUDIT.TableID = 'A' AND dbo.AUDIT.ActionType = 'I' AND
dbo.AUDIT.DataType = inserted.DataType AND
    dbo.AUDIT.SampleYear = inserted.SampleYear AND dbo.AUDIT.SpeciesID =
inserted.SpeciesID AND
```

```
    dbo.AUDIT.SequenceNumber = inserted.SequenceNumber AND dbo.AUDIT.FishNumber =
inserted.FishNumber
END

ALTER TRIGGER dbo.ANNULI_UTrig
ON dbo.ANNULI FOR UPDATE AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
IF (SELECT TrackUpdates FROM UTRIG) = 1
BEGIN
  INSERT INTO dbo.AUDIT
    SELECT 'A', 'U', DataType, SampleYear, SpeciesID, SequenceNumber, FishNumber,
USER_NAME(USER_ID()), GETDATE() FROM inserted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE dbo.AUDIT.TableID = 'A'
AND dbo.AUDIT.ActionType = 'U' AND
    dbo.AUDIT.DataType = inserted.DataType AND dbo.AUDIT.SampleYear =
inserted.SampleYear
    AND dbo.AUDIT.SpeciesID = inserted.SpeciesID AND dbo.AUDIT.SequenceNumber =
inserted.SequenceNumber
    AND dbo.AUDIT.FishNumber = inserted.FishNumber)
  UPDATE dbo.AUDIT SET UserName = USER_NAME(USER_ID()), ActionDT =
GETDATE() FROM inserted
    WHERE dbo.AUDIT.TableID = 'A' AND dbo.AUDIT.ActionType = 'U' AND
dbo.AUDIT.DataType = inserted.DataType AND
    dbo.AUDIT.SampleYear = inserted.SampleYear AND dbo.AUDIT.SpeciesID =
inserted.SpeciesID AND
    dbo.AUDIT.SequenceNumber = inserted.SequenceNumber AND dbo.AUDIT.FishNumber =
inserted.FishNumber
END

ALTER TRIGGER dbo.DOGDATA_DTrig
ON dbo.DOGDATA FOR DELETE AS
SET NOCOUNT ON
BEGIN
  INSERT INTO dbo.AUDIT
    SELECT 'D', 'D', DataType, SampleYear, SpeciesID, SequenceNumber, FishNumber,
USER_NAME(USER_ID()), GETDATE() FROM deleted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE dbo.AUDIT.TableID = 'D'
AND dbo.AUDIT.ActionType = 'D' AND
```

```
    dbo.AUDIT.DataType = deleted.DataType AND dbo.AUDIT.SampleYear =
deleted.SampleYear
    AND dbo.AUDIT.SpeciesID = deleted.SpeciesID AND dbo.AUDIT.SequenceNumber =
deleted.SequenceNumber
    AND dbo.AUDIT.FishNumber = deleted.FishNumber)
  UPDATE dbo.AUDIT SET UserName = USER_NAME(USER_ID()), ActionDT =
GETDATE() FROM deleted
    WHERE dbo.AUDIT.TableID = 'D' AND dbo.AUDIT.ActionType = 'D' AND
dbo.AUDIT.DataType = deleted.DataType AND
    dbo.AUDIT.SampleYear = deleted.SampleYear AND dbo.AUDIT.SpeciesID =
deleted.SpeciesID AND
    dbo.AUDIT.SequenceNumber = deleted.SequenceNumber AND dbo.AUDIT.FishNumber =
deleted.FishNumber
END

ALTER TRIGGER dbo.DOGDATA_ITrig
ON dbo.DOGDATA FOR INSERT AS
/* * CHANGE TRACKING */
BEGIN
  INSERT INTO dbo.AUDIT
    SELECT 'D', 'I', DataType, SampleYear, SpeciesID, SequenceNumber, FishNumber,
USER_NAME(USER_ID()), GETDATE() FROM inserted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE dbo.AUDIT.TableID = 'D'
AND dbo.AUDIT.ActionType = 'I' AND
    dbo.AUDIT.DataType = inserted.DataType AND dbo.AUDIT.SampleYear =
inserted.SampleYear
    AND dbo.AUDIT.SpeciesID = inserted.SpeciesID AND dbo.AUDIT.SequenceNumber =
inserted.SequenceNumber
    AND dbo.AUDIT.FishNumber = inserted.FishNumber)
  UPDATE dbo.AUDIT SET UserName = USER_NAME(USER_ID()), ActionDT =
GETDATE() FROM inserted
    WHERE dbo.AUDIT.TableID = 'D' AND dbo.AUDIT.ActionType = 'I' AND
dbo.AUDIT.DataType = inserted.DataType AND
    dbo.AUDIT.SampleYear = inserted.SampleYear AND dbo.AUDIT.SpeciesID =
inserted.SpeciesID AND
    dbo.AUDIT.SequenceNumber = inserted.SequenceNumber AND dbo.AUDIT.FishNumber =
inserted.FishNumber
END

ALTER TRIGGER dbo.DOGDATA_UTrig
```

```
ON dbo.DOGDATA FOR UPDATE AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
IF (SELECT TrackUpdates FROM UTRIG) = 1
BEGIN
  INSERT INTO dbo.AUDIT
    SELECT 'D', 'U', DataType, SampleYear, SpeciesID, SequenceNumber, FishNumber,
USER_NAME(USER_ID()), GETDATE() FROM inserted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE dbo.AUDIT.TableID = 'D'
AND dbo.AUDIT.ActionType = 'U' AND
    dbo.AUDIT.DataType = inserted.DataType AND dbo.AUDIT.SampleYear =
inserted.SampleYear
    AND dbo.AUDIT.SpeciesID = inserted.SpeciesID AND dbo.AUDIT.SequenceNumber =
inserted.SequenceNumber
    AND dbo.AUDIT.FishNumber = inserted.FishNumber)
  UPDATE dbo.AUDIT SET UserName = USER_NAME(USER_ID()), ActionDT =
GETDATE() FROM inserted
    WHERE dbo.AUDIT.TableID = 'D' AND dbo.AUDIT.ActionType = 'U' AND
dbo.AUDIT.DataType = inserted.DataType AND
    dbo.AUDIT.SampleYear = inserted.SampleYear AND dbo.AUDIT.SpeciesID =
inserted.SpeciesID AND
    dbo.AUDIT.SequenceNumber = inserted.SequenceNumber AND dbo.AUDIT.FishNumber =
inserted.FishNumber
END


**********************************************************************************
**********

***This stored procedure is executed daily to move change tracking information from the
AUDIT table to the
***change tracking fields in the main data tables.
**********************************************************************************
**********
CREATE PROCEDURE update_change_tracking AS

/* * DISABLE CHANGE TRACKING IN UPDATE TRIGGER */
UPDATE UTRIG SET TrackUpdates = 0

/* * UPDATE CHANGE TRACKING FIELDS */
BEGIN
```

```
SET NOCOUNT ON

UPDATE ANNULI SET Create_DFWuser_Id = AUDIT.UserName, Create_DT =
AUDIT.ActionDT, Modify_DFWuser_Id = AUDIT.UserName,
  Modify_DT = AUDIT.ActionDT
FROM ANNULI, AUDIT
WHERE ANNULI.DataType = AUDIT.DataType AND ANNULI.SampleYear =
AUDIT.SampleYear AND ANNULI.SpeciesID = AUDIT.SpeciesID
  AND ANNULI.SequenceNumber = AUDIT.SequenceNumber AND ANNULI.FishNumber =
AUDIT.FishNumber AND AUDIT.TableID ='A'
  AND AUDIT.ActionType = 'I'

UPDATE ANNULI SET Modify_DFWuser_Id = AUDIT.UserName, Modify_DT =
AUDIT.ActionDT
FROM ANNULI, AUDIT
WHERE ANNULI.DataType = AUDIT.DataType AND ANNULI.SampleYear =
AUDIT.SampleYear AND ANNULI.SpeciesID = AUDIT.SpeciesID
  AND ANNULI.SequenceNumber = AUDIT.SequenceNumber AND ANNULI.FishNumber =
AUDIT.FishNumber AND AUDIT.TableID ='A'
  AND AUDIT.ActionType = 'U'

UPDATE DOGDATA SET Create_DFWuser_Id = AUDIT.UserName, Create_DT =
AUDIT.ActionDT, Modify_DFWuser_Id = AUDIT.UserName,
   Modify_DT = AUDIT.ActionDT
FROM DOGDATA, AUDIT
WHERE DOGDATA.DataType = AUDIT.DataType AND DOGDATA.SampleYear =
AUDIT.SampleYear
   AND DOGDATA.SpeciesID = AUDIT.SpeciesID AND DOGDATA.SequenceNumber =
AUDIT.SequenceNumber
   AND DOGDATA.FishNumber = AUDIT.FishNumber AND AUDIT.TableID ='D' AND
AUDIT.ActionType = 'I'

UPDATE DOGDATA SET Modify_DFWuser_Id = AUDIT.UserName, Modify_DT =
AUDIT.ActionDT
FROM DOGDATA, AUDIT
WHERE DOGDATA.DataType = AUDIT.DataType AND DOGDATA.SampleYear =
AUDIT.SampleYear
  AND DOGDATA.SpeciesID = AUDIT.SpeciesID AND DOGDATA.SequenceNumber =
AUDIT.SequenceNumber
```

```
  AND DOGDATA.FishNumber = AUDIT.FishNumber AND AUDIT.TableID ='D' AND
AUDIT.ActionType = 'U'

UPDATE FISHDTL SET Create_DFWuser_Id = AUDIT.UserName, Create_DT =
AUDIT.ActionDT, Modify_DFWuser_Id = AUDIT.UserName,
  Modify_DT = AUDIT.ActionDT
FROM FISHDTL, AUDIT
WHERE FISHDTL.DataType = AUDIT.DataType AND FISHDTL.SampleYear =
AUDIT.SampleYear AND FISHDTL.SpeciesID = AUDIT.SpeciesID
  AND FISHDTL.SequenceNumber = AUDIT.SequenceNumber AND FISHDTL.FishNumber
= AUDIT.FishNumber
  AND AUDIT.TableID ='F' AND AUDIT.ActionType = 'I'

UPDATE FISHDTL SET Modify_DFWuser_Id = AUDIT.UserName, Modify_DT =
AUDIT.ActionDT
FROM FISHDTL, AUDIT
WHERE FISHDTL.DataType = AUDIT.DataType AND FISHDTL.SampleYear =
AUDIT.SampleYear AND FISHDTL.SpeciesID = AUDIT.SpeciesID
  AND FISHDTL.SequenceNumber = AUDIT.SequenceNumber AND FISHDTL.FishNumber
= AUDIT.FishNumber
  AND AUDIT.TableID ='F' AND AUDIT.ActionType = 'U'

UPDATE HEADER SET Create_DFWuser_Id = AUDIT.UserName, Create_DT =
AUDIT.ActionDT, Modify_DFWuser_Id = AUDIT.UserName,
  Modify_DT = AUDIT.ActionDT
FROM HEADER, AUDIT
WHERE HEADER.DataType = AUDIT.DataType AND HEADER.SampleYear =
AUDIT.SampleYear AND HEADER.SpeciesID = AUDIT.SpeciesID
  AND HEADER.SequenceNumber = AUDIT.SequenceNumber AND AUDIT.TableID ='H'
AND AUDIT.ActionType = 'I'

UPDATE HEADER SET Modify_DFWuser_Id = AUDIT.UserName, Modify_DT =
AUDIT.ActionDT
FROM HEADER, AUDIT
WHERE HEADER.DataType = AUDIT.DataType AND HEADER.SampleYear =
AUDIT.SampleYear AND HEADER.SpeciesID = AUDIT.SpeciesID
  AND HEADER.SequenceNumber = AUDIT.SequenceNumber AND AUDIT.TableID ='H'
AND AUDIT.ActionType = 'U'

DELETE FROM AUDIT WHERE ActionType <> 'D'
```

```
SET NOCOUNT OFF
END

/* * ENABLE CHANGE TRACKING IN UPDATE TRIGGER */
UPDATE UTRIG SET TrackUpdates = 1

GRANT EXECUTE ON update_change_tracking TO dbo
```

# Appendix P: CTLS Main Table Definitions

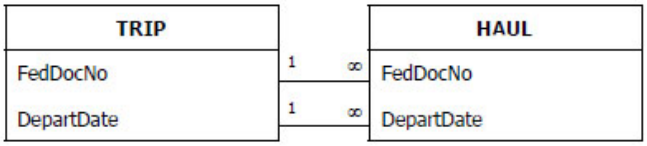
TRIP Table:

**Columns**

| Name | Type | Size |
|---|---|---|
| FedDocNo | Text | 6 |
| DepartDate | Date/Time | 8 |
| DepartTime | Integer | 2 |
| DepartPort | Text | 3 |
| ReturnDate | Date/Time | 8 |
| ReturnTime | Integer | 2 |
| ReturnPort | Text | 3 |
| DaysOut | Integer | 2 |
| DaysFished | Integer | 2 |
| CrewSize | Integer | 2 |
| TripType | Integer | 2 |
| NoOfTows | Integer | 2 |
| FirstBuyer | Text | 15 |
| FirstTicketNo | Text | 7 |
| SecondBuyer | Text | 15 |
| SecondTicketNo | Text | 7 |
| ThirdBuyer | Text | 15 |
| ThirdTicketNo | Text | 7 |
| Comment | Text | 100 |
| Create_DFWuser_id | Text | 12 |
| Create_DT | Date/Time | 8 |
| Modify_DFWuser_id | Text | 12 |
| Modify_DT | Date/Time | 8 |

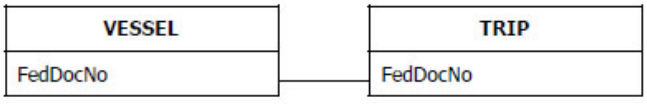
**Relationships**

**TRIPHAUL**




| | |
|---|---|
| Attributes: | Enforced, Cascade Updates, Cascade Deletes, Left Join |
| RelationshipType: | One-To-Many |

**VESSELTRIP**




| | |
|---|---|
| Attributes: | Not Enforced |
| RelationshipType: | One-To-Many |

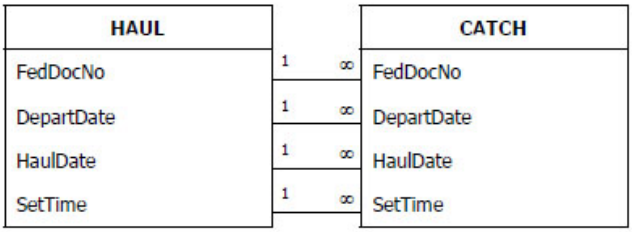
HAUL Table:

**Columns**

| Name | Type | Size |
|---|---|---|
| FedDocNo | Text | 6 |
| DepartDate | Date/Time | 8 |
| HaulDate | Date/Time | 8 |
| SetTime | Integer | 2 |
| UpTime | Integer | 2 |
| TowHours | Integer | 2 |
| TowMinutes | Integer | 2 |
| SetLatDeg | Integer | 2 |
| SetLatMin | Double | 8 |
| SetLonDeg | Integer | 2 |
| SetLonMin | Double | 8 |
| UpLatDeg | Integer | 2 |
| UpLatMin | Double | 8 |
| UpLonDeg | Integer | 2 |
| UpLonMin | Double | 8 |
| MFMgmtArea | Integer | 2 |
| WDFArea | Integer | 2 |
| GroundCode | Text | 2 |
| PMFCArea | Text | 2 |
| AvgDepth | Integer | 2 |
| NetType | Text | 1 |
| TargetStrategy | Text | 4 |
| Create_DFWuser_id | Text | 12 |
| Create_DT | Date/Time | 8 |
| Modify_DFWuser_id | Text | 12 |
| Modify_DT | Date/Time | 8 |

**Relationships**

**HAULCATCH**




Attributes: Enforced, Cascade Updates, Cascade Deletes, Left Join
RelationshipType: One-To-Many

**TRIPHAUL**


Attributes: Enforced, Cascade Updates, Cascade Deletes, Left Join
RelationshipType: One-To-Many

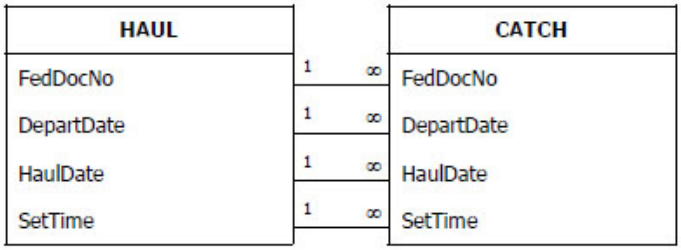
CATCH Table:

**Columns**

| Name | Type | Size |
|---|---|---|
| FedDocNo | Text | 6 |
| DepartDate | Date/Time | 8 |
| HaulDate | Date/Time | 8 |
| SetTime | Integer | 2 |
| SpeciesCode | Text | 4 |
| PoundsCatch | Long Integer | 4 |
| Create_DFWuser_id | Text | 12 |
| Create_DT | Date/Time | 8 |
| Modify_DFWuser_id | Text | 12 |
| Modify_DT | Date/Time | 8 |

**Relationships**

**HAULCATCH**




| | |
|---|---|
| Attributes: | Enforced, Cascade Updates, Cascade Deletes, Left Join |
| RelationshipType: | One-To-Many |

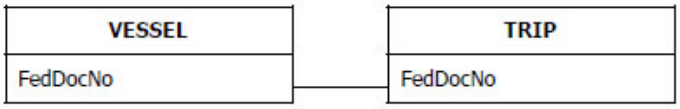
VESSELTable:

**Columns**

| Name | Type | Size |
|---|---|---|
| FedDocNo | Text | 6 |
| VesselName | Text | 20 |
| WARegNo | Long Integer | 4 |

**Relationships**

**VESSELTRIP**




| | |
|---|---|
| Attributes: | Not Enforced |
| RelationshipType: | One-To-Many |

AUDIT Table:

**Columns**

| Name | Type | Size |
|---|---|---|
| TableID | Text | 1 |
| ActionType | Text | 1 |
| FedDocNo | Text | 6 |
| DepartDate | Date/Time | 8 |
| HaulDate | Date/Time | 8 |
| SetTime | Integer | 2 |
| SpeciesCode | Text | 4 |
| UserName | Text | 12 |
| ActionDT | Date/Time | 8 |

# Appendix Q: CTLS Code List Look-Up Tables

MARKET:

| PacFINCode | Description |
|---|---|
| ARTH | Arrowtooth Flounder |
| BANK | Bank Rockfish |
| BCAC | Bacacio |
| BSOL | Butter Sole |
| BSPT | Blackspotted Rockfish |
| CABZ | Cabezon |
| CFSO | Curlfin Sole |
| CHLB | California Halibut |
| CLPR | Chilipepper |
| CNRY | Canary Rockfish |
| DBRK | Darkblotched Rockfish |
| DOVR | Dover Sole |
| DSRK | Dogfish |
| EGLS | English Sole |
| GRDR | Unspecified Grenadier |
| GSTG | Green Sturgeon |
| JMCK | Jack Mackerel |
| LCOD | Lingcod |
| LSPN | Longspine Thornyhead |
| NUSF | Shelf Rockfish |
| NUSP | Slope Rockfish |
| NUSR | Near-shore Rockfish |
| OCTO | Octopus |
| OFLT | Other Flatfish |
| OSRK | Other Shark |
| PCOD | Pacific Cod |
| PLCK | Walleye Pollack |
| PMCK | Pacific Mackerel |
| POP | Pacific Ocean Perch |
| PTRL | Petrale Sole |
| PWHT | Pacific Whiting (Hake) |
| REX | Rex Sole |
| RSOL | Rock Sole |
| SABL | Sablefish |
| SBLY | Shortbelly Rockfish |
| SDAB | Sanddabs |
| SKAT | All Skates and Rays |
| SSOL | Sand Sole |
| SSPN | Shortspine Thornyhead |
| SSRK | Soupfin Shark |
| STRY | Starry Flounder |
| UMCK | Unspecified Mackerel |
| WDOW | Widow Rockfish |
| WEEL | Wolf Eel |
| WSTG | White Sturgeon |
| YEYE | Yelloweye Rockfish |
| YTRK | Yellowtail Rockfish |

TARGET:

| PacFINCode | Description |
|---|---|
| ARTH | Arrowtooth Flounder |
| BRSH | Bottom Rockfish - Shelf |
| BRSL | Bottom Rockfish - Slope |
| DOVR | Dover Sole |
| DSRK | Dogfish |
| DTS | Dover/Thornyheads/Sable |
| DWD | Deepwater Dover |
| EGLS | English Sole |
| NSM | Nearshore Mix |
| PCOD | Pacific Cod |
| POP | Pacific Ocean Perch |
| PTRL | Petrale Sole |
| PWHT | Pacific Whiting (Hake) |
| RSOL | Rock Sole |
| SABL | Sablefish |
| STRY | Starry Flounder |
| THHD | Thornyheads (Mixed) |

VESSEL:

| FedDocNo | VesselName | WARegNo |
|---|---|---|
| 209613 | Oregonian | 20433 |
| 209773 | Marie Ann Gail | 10775 |
| 212958 | New York | 360 |
| 214795 | Panther | 110 |
| 215249 | Columbia | 4346 |
| 219888 | Golden North | 3449 |
| 220086 | Lily Marlene | 18555 |
| 222358 | Tulip | 347 |
| 226380 | Sunlight | 1458 |
| 228687 | Orbit | 715 |
| 235851 | Arrow IV | 15046 |
| 236389 | New Washington | 305 |
| 245569 | Sara Frances | 19080 |
| 245978 | Margaret E | 11486 |
| 246396 | Brookfield | 24 |
| 246957 | Dakota | 11275 |
| 247438 | Cygnet II | 11991 |
| 249330 | Nestucca | 13761 |
| 249564 | Pacific Queen | 15875 |
| 254713 | Lucky Strike | 10675 |
| 276600 | Thrush | 11230 |
| 503972 | Mi-Lo | 1171 |
| 505917 | Billie Jean | 21383 |
| 509492 | Pam Bay | 31583 |
| 509552 | Mark I | 8340 |
| 512179 | Orion | 3799 |
| 514551 | Peggy Marie | 19431 |
| 514665 | Grumpy J | 20282 |
| 515274 | Windjammer | 142 |
| 516627 | Miss Mary | 17834 |
| 516881 | St Janet | 32245 |
| 518937 | Pacific Challenger | 10991 |

GEAR:

| Gear | Description | AgencyCode | PacFINCode |
|---|---|---|---|
| B | Bottom Trawl (Small Footrope) | 36 | GFS |
| F | Selective Flatfish | 37 | FTS |
| L | Set Line | 43 | LGL |
| M | Midwater Trawl | 34 | MDT |
| R | Roller Trawl | 35 | RLT |

TRIPTYPE:

| TripType | Description |
|---|---|
| 1 | Canada |
| 2 | Alaska |
| 3 | Coastal |
| 4 | Strait & Gulf |
| 5 | Puget Sound & Hood Canal |

PORT:

| MFPortCode | Description | WDFPortCode | MFPortCluster | PacFINCode |
|---|---|---|---|---|
| ABE | Aberdeen | 210 | GRH | GRH |
| ALA | Alaska (all ports) | 501 | ALA | ALA |
| ANA | Anacortes | 105 | ANA | ANA |
| B.C | Bay Center | 310 | WLB | WLB |
| BEL | Bellingham | 110 | BEL | BLL |
| BLA | Blaine | 115 | BLA | BLN |
| BRE | Bremerton | 120 | PSD | OSP |
| BRI | Brinnon | 181 | PSD | OWA |
| C.B | Copalis Beach | 230 | GRH | CPL |
| CAL | California (all ports) | 502 | CAL | CAL |
| CHI | Chinook | 409 | COL | LWC |
| COU | Coupeville | 125 | PSD | ONP |
| EVE | Everett | 130 | PSD | EVR |
| F.H | Friday Harbor | 135 | BEL | FRI |
| HOQ | Hoquiam | 240 | GRH | GRH |
| ILW | Ilwaco | 421 | COL | LWC |
| KEL | Kelso | 423 | COL | OCR |
| L.B | Long Beach | 320 | WLB | WLB |
| L.C | La Conner | 140 | ANA | LAC |
| L.P | La Push | 260 | GRH | LAP |
| LAC | La Conner | 140 | ANA | LAC |
| N.B | Neah Bay | 150 | N.B | NEA |
| NAS | Naselle | 340 | WLB | WLB |
| OLY | Olympia | 155 | PSD | OLY |
| ORE | Oregon (all ports) | 503 | ORE | ORE |
| P.A | Port Angeles | 160 | PEN | PAG |
| P.R | Point Roberts | 156 | BLA | ONP |
| P.T | Port Townsend | 165 | PEN | TNS |
| POU | Poulsbo | 169 | PSD | OSP |
| RAY | Raymond | 345 | WLB | WLB |
| S.B | South Bend | 350 | WLB | WLB |
| SEA | Seattle | 170 | PSD | SEA |
| SEQ | Sequim | 175 | PEN | SEQ |
| SHE | Shelton | 180 | PSD | SHL |
| TAC | Tacoma | 190 | PSD | TAC |
| TAH | Taholah | 290 | GRH | OWC |
| TOK | Tokeland | 360 | WLB | WLB |
| VAN | Vancouver (Wash.) | 438 | COL | OCR |
| WES | Westport | 295 | GRH | WPT |

AREA:

| WDFArea |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |
| 13 |
| 14 |
| 15 |
| 16 |
| 17 |
| 20 |
| 21 |
| 22 |
| 23 |
| 24 |
| 25 |
| 26 |
| 27 |
| 28 |
| 29 |
| 30 |
| 31 |
| 32 |
| 33 |
| 34 |
| 36 |
| 38 |
| 40 |
| 42 |
| 48 |
| 49 |
| 50 |
| 51 |
| 52 |
| 53 |
| 54 |
| 55 |
| 83 |
| 99 |

GROUND:

| GroundCode |
|---|
| 1 |
| 10 |
| 11 |
| 12 |
| 13 |
| 14 |
| 15 |
| 16 |
| 2 |
| 20 |
| 21 |
| 22 |
| 23 |
| 24 |
| 25 |
| 26 |
| 3 |
| 30 |
| 31 |
| 32 |
| 33 |
| 4 |
| 40 |
| 41 |
| 42 |
| 43 |
| 44 |
| 5 |
| 50 |
| 51 |
| A |
| A0 |
| A1 |
| A2 |
| A3 |
| A4 |
| A5 |
| A6 |
| B |
| B0 |
| B1 |
| B2 |
| B3 |
| B4 |
| B5 |
| B6 |
| C |
| C0 |
| C1 |
| C2 |
| C3 |
| D |
| D0 |
| D1 |
| D2 |
| D3 |
| D4 |

# Appendix R: CTLS Change Tracking Triggers and Update Procedure

```
**********************************************************************************************************
***This code must be appended to existing code for cascade delete, cascade update, and
referential integrity.
**********************************************************************************************************
ALTER TRIGGER dbo.T_TRIP_DTrig
ON dbo.TRIP FOR DELETE AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
BEGIN
  INSERT INTO AUDIT
    SELECT 'T', 'D', FedDocNo, DepartDate, '', 0, '', USER_NAME(USER_ID()), GETDATE()
FROM deleted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE AUDIT.TableID = 'T' AND
AUDIT.ActionType = 'D' AND
    AUDIT.FedDocNo = deleted.FedDocNo AND AUDIT.DepartDate = deleted.DepartDate)
  UPDATE AUDIT SET AUDIT.UserName = USER_NAME(USER_ID()), AUDIT.ActionDT
= GETDATE() FROM deleted
    WHERE  AUDIT.TableID = 'T' AND AUDIT.ActionType = 'D' AND AUDIT.FedDocNo =
deleted.FedDocNo AND
    AUDIT.DepartDate = deleted.DepartDate
END

ALTER TRIGGER dbo.T_TRIP_ITrig
ON dbo.TRIP FOR INSERT AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
BEGIN
  INSERT INTO AUDIT
    SELECT 'T', 'I', FedDocNo, DepartDate, '', 0, '', USER_NAME(USER_ID()), GETDATE()
FROM inserted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE AUDIT.TableID = 'T' AND
AUDIT.ActionType = 'I' AND
    AUDIT.FedDocNo = inserted.FedDocNo AND AUDIT.DepartDate = inserted.DepartDate)
  UPDATE AUDIT SET AUDIT.UserName = USER_NAME(USER_ID()), AUDIT.ActionDT
= GETDATE() FROM inserted
```

```
    WHERE  AUDIT.TableID = 'T' AND AUDIT.ActionType = 'I' AND AUDIT.FedDocNo =
inserted.FedDocNo AND
    AUDIT.DepartDate = inserted.DepartDate
END

ALTER TRIGGER dbo.T_TRIP_UTrig
ON dbo.TRIP FOR UPDATE AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
IF (SELECT TrackUpdates FROM UTRIG) = 1
BEGIN
  INSERT INTO AUDIT
    SELECT 'T', 'U', FedDocNo, DepartDate, '', 0, '', USER_NAME(USER_ID()), GETDATE()
FROM inserted
    WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE AUDIT.TableID = 'T' AND
AUDIT.ActionType = 'U' AND
    AUDIT.FedDocNo = inserted.FedDocNo AND AUDIT.DepartDate = inserted.DepartDate)
  UPDATE AUDIT SET AUDIT.UserName = USER_NAME(USER_ID()), AUDIT.ActionDT
= GETDATE() FROM inserted
    WHERE  AUDIT.TableID = 'T' AND AUDIT.ActionType = 'U' AND AUDIT.FedDocNo =
inserted.FedDocNo AND
    AUDIT.DepartDate = inserted.DepartDate
END

ALTER TRIGGER dbo.T_HAUL_DTrig
ON dbo.HAUL FOR DELETE AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
BEGIN
  INSERT INTO AUDIT
    SELECT 'H', 'D', FedDocNo, DepartDate, HaulDate, SetTime, '',
USER_NAME(USER_ID()), GETDATE() FROM
    deleted WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE AUDIT.TableID = 'H'
AND AUDIT.ActionType = 'D'
    AND AUDIT.FedDocNo = deleted.FedDocNo AND AUDIT.DepartDate =
deleted.DepartDate AND
    AUDIT.HaulDate = deleted.HaulDate AND AUDIT.SetTime = deleted.SetTime)
  UPDATE AUDIT SET AUDIT.UserName = USER_NAME(USER_ID()), AUDIT.ActionDT
= GETDATE() FROM deleted
```

```
    WHERE  AUDIT.TableID = 'H' AND AUDIT.ActionType = 'D' AND AUDIT.FedDocNo =
deleted.FedDocNo AND
    AUDIT.DepartDate = deleted.DepartDate AND AUDIT.HaulDate = deleted.HaulDate AND
    AUDIT.SetTime = deleted.SetTime
END

ALTER TRIGGER dbo.T_HAUL_ITrig
ON dbo.HAUL FOR INSERT AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
BEGIN
  INSERT INTO AUDIT
    SELECT 'H', 'I', FedDocNo, DepartDate, HaulDate, SetTime, '', USER_NAME(USER_ID()),
GETDATE() FROM
    inserted WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE AUDIT.TableID = 'H'
AND AUDIT.ActionType = 'I'
    AND AUDIT.FedDocNo = inserted.FedDocNo AND AUDIT.DepartDate =
inserted.DepartDate AND
    AUDIT.HaulDate = inserted.HaulDate AND AUDIT.SetTime = inserted.SetTime)
  UPDATE AUDIT SET AUDIT.UserName = USER_NAME(USER_ID()), AUDIT.ActionDT
= GETDATE() FROM inserted
    WHERE  AUDIT.TableID = 'H' AND AUDIT.ActionType = 'I' AND AUDIT.FedDocNo =
inserted.FedDocNo AND
    AUDIT.DepartDate = inserted.DepartDate AND AUDIT.HaulDate = inserted.HaulDate
AND
    AUDIT.SetTime = inserted.SetTime
END

ALTER TRIGGER dbo.T_HAUL_UTrig
ON dbo.HAUL FOR UPDATE AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
IF (SELECT TrackUpdates FROM UTRIG) = 1
BEGIN
  INSERT INTO AUDIT
    SELECT 'H', 'U', FedDocNo, DepartDate, HaulDate, SetTime, '',
USER_NAME(USER_ID()), GETDATE() FROM
    inserted WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE AUDIT.TableID = 'H'
AND AUDIT.ActionType = 'U'
```

```
    AND AUDIT.FedDocNo = inserted.FedDocNo AND AUDIT.DepartDate =
inserted.DepartDate AND
    AUDIT.HaulDate = inserted.HaulDate AND AUDIT.SetTime = inserted.SetTime)
  UPDATE AUDIT SET AUDIT.UserName = USER_NAME(USER_ID()), AUDIT.ActionDT
= GETDATE() FROM inserted
    WHERE  AUDIT.TableID = 'H' AND AUDIT.ActionType = 'U' AND AUDIT.FedDocNo =
inserted.FedDocNo AND
    AUDIT.DepartDate = inserted.DepartDate AND AUDIT.HaulDate = inserted.HaulDate
AND
    AUDIT.SetTime = inserted.SetTime
END

ALTER TRIGGER dbo.T_CATCH_DTrig
ON dbo.CATCH FOR DELETE AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
BEGIN
  INSERT INTO AUDIT
    SELECT 'C', 'D', FedDocNo, DepartDate, HaulDate, SetTime, SpeciesCode,
USER_NAME(USER_ID()), GETDATE() FROM
    deleted WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE AUDIT.TableID = 'C'
AND AUDIT.ActionType = 'D'
    AND AUDIT.FedDocNo = deleted.FedDocNo AND AUDIT.DepartDate =
deleted.DepartDate AND
    AUDIT.HaulDate = deleted.HaulDate AND AUDIT.SetTime = deleted.SetTime AND
AUDIT.SpeciesCode = deleted.SpeciesCode)
  UPDATE AUDIT SET AUDIT.UserName = USER_NAME(USER_ID()), AUDIT.ActionDT
= GETDATE() FROM deleted
    WHERE  AUDIT.TableID = 'C' AND AUDIT.ActionType = 'D' AND AUDIT.FedDocNo =
deleted.FedDocNo AND
    AUDIT.DepartDate = deleted.DepartDate AND AUDIT.HaulDate = deleted.HaulDate AND
    AUDIT.SetTime = deleted.SetTime AND AUDIT.SpeciesCode = deleted.SpeciesCode
END

ALTER TRIGGER dbo.T_CATCH_ITrig
ON dbo.CATCH FOR INSERT AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
BEGIN
  INSERT INTO AUDIT
```

```
    SELECT 'C', 'I', FedDocNo, DepartDate, HaulDate, SetTime, SpeciesCode,
USER_NAME(USER_ID()), GETDATE() FROM
    inserted WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE AUDIT.TableID = 'C'
AND AUDIT.ActionType = 'I'
    AND AUDIT.FedDocNo = inserted.FedDocNo AND AUDIT.DepartDate =
inserted.DepartDate AND
    AUDIT.HaulDate = inserted.HaulDate AND AUDIT.SetTime = inserted.SetTime AND
AUDIT.SpeciesCode = inserted.SpeciesCode)
  UPDATE AUDIT SET AUDIT.UserName = USER_NAME(USER_ID()), AUDIT.ActionDT
= GETDATE() FROM inserted
    WHERE  AUDIT.TableID = 'C' AND AUDIT.ActionType = 'I' AND AUDIT.FedDocNo =
inserted.FedDocNo AND
    AUDIT.DepartDate = inserted.DepartDate AND AUDIT.HaulDate = inserted.HaulDate
AND
    AUDIT.SetTime = inserted.SetTime AND AUDIT.SpeciesCode = inserted.SpeciesCode
END

ALTER TRIGGER dbo.T_CATCH_UTrig
ON dbo.CATCH FOR UPDATE AS
SET NOCOUNT ON
/* * CHANGE TRACKING */
IF (SELECT TrackUpdates FROM UTRIG) = 1
BEGIN
  INSERT INTO AUDIT
    SELECT 'C', 'U', FedDocNo, DepartDate, HaulDate, SetTime, SpeciesCode,
USER_NAME(USER_ID()), GETDATE() FROM
    inserted WHERE NOT EXISTS (SELECT * FROM AUDIT WHERE AUDIT.TableID = 'C'
AND AUDIT.ActionType = 'U'
    AND AUDIT.FedDocNo = inserted.FedDocNo AND AUDIT.DepartDate =
inserted.DepartDate AND
    AUDIT.HaulDate = inserted.HaulDate AND AUDIT.SetTime = inserted.SetTime AND
AUDIT.SpeciesCode = inserted.SpeciesCode)
  UPDATE AUDIT SET AUDIT.UserName = USER_NAME(USER_ID()), AUDIT.ActionDT
= GETDATE() FROM inserted
    WHERE  AUDIT.TableID = 'C' AND AUDIT.ActionType = 'U' AND AUDIT.FedDocNo =
inserted.FedDocNo AND
    AUDIT.DepartDate = inserted.DepartDate AND AUDIT.HaulDate = inserted.HaulDate
AND
    AUDIT.SetTime = inserted.SetTime AND AUDIT.SpeciesCode = inserted.SpeciesCode
END
```

```
**********************************************************************************
******************
***This stored procedure is executed daily to move change tracking information from the
AUDIT table to the
***change tracking fields in the main data tables.
**********************************************************************************
******************

CREATE PROCEDURE update_change_tracking AS

/* * DISABLE CHANGE TRACKING IN UPDATE TRIGGER */
UPDATE UTRIG SET TrackUpdates = 0

/* * UPDATE CHANGE TRACKING FIELDS */
BEGIN
SET NOCOUNT ON

UPDATE CATCH SET Create_DFWuser_id = AUDIT.UserName, Create_DT =
AUDIT.ActionDT, Modify_DFWuser_id = AUDIT.UserName, Modify_DT =
AUDIT.ActionDT
FROM CATCH, AUDIT
WHERE CATCH.FedDocNo = AUDIT.FedDocNo AND CATCH.DepartDate =
AUDIT.DepartDate AND CATCH.HaulDate = AUDIT.HaulDate AND CATCH.SetTime =
AUDIT.SetTime
  AND CATCH.SpeciesCode = AUDIT.SpeciesCode AND AUDIT.TableID = 'C' AND
AUDIT.ActionType = 'I'

UPDATE CATCH SET Modify_DFWuser_id = AUDIT.UserName, Modify_DT =
AUDIT.ActionDT
FROM CATCH, AUDIT
WHERE CATCH.FedDocNo = AUDIT.FedDocNo AND CATCH.DepartDate =
AUDIT.DepartDate AND CATCH.HaulDate = AUDIT.HaulDate AND CATCH.SetTime =
AUDIT.SetTime
  AND CATCH.SpeciesCode = AUDIT.SpeciesCode AND AUDIT.TableID = 'C' AND
AUDIT.ActionType = 'U'

UPDATE HAUL SET Create_DFWuser_id = AUDIT.UserName, Create_DT =
AUDIT.ActionDT, Modify_DFWuser_id = AUDIT.UserName, Modify_DT =
AUDIT.ActionDT
FROM HAUL, AUDIT
```

```
WHERE HAUL.FedDocNo = AUDIT.FedDocNo AND HAUL.DepartDate =
AUDIT.DepartDate AND HAUL.HaulDate = AUDIT.HaulDate AND HAUL.SetTime =
AUDIT.SetTime
  AND AUDIT.TableID = 'H' AND AUDIT.ActionType = 'I'

UPDATE HAUL SET Modify_DFWuser_id = AUDIT.UserName, Modify_DT =
AUDIT.ActionDT
FROM HAUL, AUDIT
WHERE HAUL.FedDocNo = AUDIT.FedDocNo AND HAUL.DepartDate =
AUDIT.DepartDate AND HAUL.HaulDate = AUDIT.HaulDate AND  HAUL.SetTime =
AUDIT.SetTime
  AND AUDIT.TableID = 'H' AND AUDIT.ActionType = 'U'

UPDATE TRIP SET Create_DFWuser_id = AUDIT.UserName, Create_DT =
AUDIT.ActionDT, Modify_DFWuser_id = AUDIT.UserName, Modify_DT =
AUDIT.ActionDT
FROM TRIP, AUDIT
WHERE TRIP.FedDocNo = AUDIT.FedDocNo AND TRIP.DepartDate = AUDIT.DepartDate
AND AUDIT.TableID = 'T'
AND AUDIT.ActionType = 'I'

UPDATE TRIP SET Modify_DFWuser_id = AUDIT.UserName, Modify_DT =
AUDIT.ActionDT
FROM TRIP, AUDIT
WHERE TRIP.FedDocNo = AUDIT.FedDocNo AND TRIP.DepartDate = AUDIT.DepartDate
AND AUDIT.TableID = 'T'
AND AUDIT.ActionType = 'U'

DELETE FROM AUDIT WHERE ActionType <> 'D'

SET NOCOUNT OFF
END

/* * ENABLE CHANGE TRACKING IN UPDATE TRIGGER */
UPDATE UTRIG SET TrackUpdates = 1

GRANT EXECUTE ON update_change_tracking TO dbo
```

# Appendix S: Fill Missing Data for CTLS Stored Procedure

```
CREATE PROCEDURE fill_missing_data AS

/* * DISABLE CHANGE TRACKING IN UPDATE TRIGGER */
UPDATE UTRIG SET TrackUpdates = 0

/* * USE DATA ENTERED TO POPULATE OTHER FIELDS */
BEGIN
SET NOCOUNT ON

/* * FILL DAYS FISHED AND NUMBER OF TOWS */
CREATE TABLE #haulcount_tmp
(
FedDocNo varchar(6),
DepartDate datetime,
DaysFished smallint,
NoOfTows smallint
)

INSERT INTO #haulcount_tmp (FedDocNo, DepartDate, DaysFished, NoOfTows)
SELECT FedDocNo, DepartDate, COUNT(DISTINCT HaulDate), COUNT(HaulDate)
FROM HAUL
GROUP BY FedDocNo, DepartDate
ORDER BY FedDocNo, DepartDate

UPDATE TRIP SET TRIP.DaysFished = #haulcount_tmp.DaysFished, TRIP.NoOfTows =
#haulcount_tmp.NoOfTows
FROM TRIP, #haulcount_tmp
WHERE TRIP.FedDocNo = #haulcount_tmp.FedDocNo AND TRIP.DepartDate =
#haulcount_tmp.DepartDate

/* * FILL TOW TIME */
CREATE TABLE #towtime_tmp
(
FedDocNo varchar(6),
DepartDate datetime,
HaulDate datetime,
SetTime smallint,
SetMinutes smallint,
```

```
UpMinutes smallint,
MinDiff smallint
)

INSERT INTO #towtime_tmp (FedDocNo, DepartDate, HaulDate, SetTime, SetMinutes,
UpMinutes)
SELECT FedDocNo, DepartDate, HaulDate, SetTime,
  FLOOR(SetTime / 100) * 60 + (SetTime % 100) as SetMinutes,
  FLOOR(UpTime / 100) * 60 + (UpTime % 100) as UpMinutes
FROM HAUL

UPDATE #towtime_tmp
  SET MinDiff =
    CASE
      WHEN UpMinutes < SetMinutes
        THEN 1440 + UpMinutes - SetMinutes
      ELSE
        UpMinutes - SetMinutes
    END

UPDATE HAUL
  SET HAUL.TowHours = FLOOR(#towtime_tmp.MinDiff / 60),
    HAUL.TowMinutes = #towtime_tmp.MinDiff % 60
FROM HAUL, #towtime_tmp
WHERE HAUL.FedDocNo = #towtime_tmp.FedDocNo AND HAUL.DepartDate =
#towtime_tmp.DepartDate
   AND HAUL.HaulDate = #towtime_tmp.HaulDate AND HAUL.SetTime =
#towtime_tmp.SetTime

/* * FILL WDF AREA AND GROUND CODE FROM LAT/LON VALUES */
CREATE TABLE #wdfarea_tmp
(
FedDocNo varchar(6),
DepartDate datetime,
HaulDate datetime,
SetTime smallint,
SetLatInt smallint,
SetLonInt smallint,
SetLatDec float,
SetLonDec float,
```

```
WDFArea smallint,
GroundCode varchar(2)
)

INSERT INTO #wdfarea_tmp (FedDocNo, DepartDate, HaulDate, SetTime, SetLatInt,
SetLonInt, SetLatDec, SetLonDec)
SELECT FedDocNo, DepartDate, HaulDate, SetTime,
  SetLatDeg*100 + FLOOR(SetLatMin + 0.5),
  SetLonDeg*100 + FLOOR(SetLonMin + 0.5),
  SetLatDeg + SetLatMin/60,
  SetLonDeg + SetLonMin/60
FROM HAUL
WHERE SetLatDeg > 0 AND SetLonDeg > 0

UPDATE #wdfarea_tmp
  SET WDFArea =
    CASE
      WHEN SetLatInt < 3800 OR SetLatInt > 7000
        THEN NULL
      WHEN SetLatInt > 4700 AND SetLonInt < 12328
        THEN NULL
      WHEN SetLatInt > 4807 AND SetLonInt < 12424
        THEN 23
      WHEN SetLatInt > 4807 AND SetLonInt < 12438
        THEN 29
      WHEN SetLatInt >= 4823 AND SetLonInt < 12444
        THEN 29
      WHEN SetLatInt <= 4546
        THEN
          CASE
            WHEN SetLatInt > 4418
              THEN 32
            WHEN SetLatInt > 4250
              THEN 34
            WHEN SetLatInt > 4200
              THEN 36
            WHEN SetLatInt > 4030
              THEN 38
            WHEN SetLatInt > 3800
              THEN 40
```

```
      ELSE 42
    END
WHEN SetLatInt > 4826
 THEN
  CASE
   WHEN SetLonInt > 13700
    THEN
     CASE
      WHEN SetLonInt <= 14700
       THEN 54
      WHEN SetLonInt <= 15400
       THEN 53
      WHEN SetLonInt <= 17000
       THEN
        CASE
         WHEN SetLatDec > 115.71 - 0.372 * SetLonDec
          THEN 48
         WHEN SetLonInt <= 15900
          THEN 52
         ELSE 51
        END
      WHEN SetLatInt <= 5500
       THEN 50
      ELSE 49
     END
   ELSE
    CASE
     WHEN SetLatInt <= 4900
      THEN
       CASE
        WHEN SetLonDec <= 185.203 - 1.247 * SetLatDec
         THEN 12
        WHEN SetLatDec <= 48.5 AND SetLonDec <= 256 - 2.7 * SetLatDec
         THEN 13
        WHEN SetLonInt <= 12515
         THEN 11
        WHEN SetLatInt <= 4843
         THEN 10
        ELSE 8
       END
```

```
        WHEN SetLatInt <= 4928
          THEN
            CASE
              WHEN SetLonDec > 153.8 - 0.552 * SetLatDec
                THEN 6
              ELSE 7
            END
        WHEN SetLatInt <= 5000
          THEN 5
        WHEN SetLatInt <= 5030
          THEN 4
        WHEN SetLatInt <= 5115
          THEN 3
        WHEN SetLatInt <= 5210
          THEN
            CASE
              WHEN SetLonInt <= 13100 OR SetLatInt <= 5200
                THEN 2
              ELSE 31
            END
        WHEN SetLatInt <= 5415
          THEN
            CASE
              WHEN SetLonDec > 85.09 + 0.885 * SetLatDec
                THEN 31
              WHEN SetLatDec <= 80.66 - 0.208 * SetLonDec
                THEN 1
              ELSE 99
            END
        WHEN SetLatInt <= 5440
          THEN
            CASE
              WHEN SetLonInt > 13300
                THEN 31
              ELSE 99
            END
        ELSE 55
      END
    END
  ELSE
```

```
      CASE
       WHEN SetLatInt <= 4616
        THEN 30
       WHEN SetLatInt <= 4650
        THEN 17
       WHEN SetLatInt <= 4720
        THEN 16
       ELSE
        CASE
         WHEN SetLonDec > 256 - 2.7 * SetLatDec
          THEN 9
         WHEN SetLonDec > 185.203 - 1.247 * SetLatDec
          THEN 13
         WHEN SetLatInt <= 4745
          THEN 15
         WHEN SetLatInt <= 4800
          THEN 14
         ELSE 12
        END
      END
   END,
  GroundCode =
   CASE
    WHEN SetLatInt < 3800 OR SetLatInt > 7000
     THEN NULL
    WHEN SetLatInt > 4700 AND SetLonInt < 12328
     THEN NULL
    WHEN SetLatInt > 4807 AND SetLonInt < 12344
     THEN 'C1'
    WHEN SetLatInt > 4807 AND SetLonInt < 12424
     THEN 'C2'
   END

UPDATE HAUL SET HAUL.WDFArea =#wdfarea_tmp.WDFArea
FROM HAUL, #wdfarea_tmp
WHERE HAUL.FedDocNo = #wdfarea_tmp.FedDocNo AND HAUL.DepartDate =
#wdfarea_tmp.DepartDate
  AND HAUL.HaulDate = #wdfarea_tmp.HaulDate AND HAUL.SetTime =
#wdfarea_tmp.SetTime AND #wdfarea_tmp.WDFArea > 0
```

```
UPDATE HAUL SET HAUL.GroundCode =#wdfarea_tmp.GroundCode
FROM HAUL, #wdfarea_tmp
WHERE HAUL.FedDocNo = #wdfarea_tmp.FedDocNo AND HAUL.DepartDate =
#wdfarea_tmp.DepartDate
  AND HAUL.HaulDate = #wdfarea_tmp.HaulDate AND HAUL.SetTime =
#wdfarea_tmp.SetTime
  AND #wdfarea_tmp.GroundCode IS NOT NULL

/* * FILL MARINE FISH MANAGEMENT AREA AND PMFC AREA FROM WDF AREA */
UPDATE HAUL
  SET MFMgmtArea =
    CASE
      WHEN WDFArea = 40
        THEN 0
      WHEN WDFArea IN (48, 49)
        THEN 50
      WHEN WDFArea = 33
        THEN 55
      WHEN WDFArea IN (1, 2, 3, 31, 99)
        THEN 56
      WHEN WDFArea IN (4, 5, 6, 7, 8, 10, 11)
        THEN 57
      WHEN WDFArea IN (9, 13)
        THEN 58
      WHEN WDFArea IN (12, 14, 15)
        THEN 59
      WHEN WDFArea IN (16, 17)
        THEN 60
      WHEN WDFArea IN (30, 32, 34, 36)
        THEN 61
      WHEN WDFArea = 38
        THEN 62
      WHEN WDFArea = 42
        THEN 63
      WHEN WDFArea IN (20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 50, 51, 52, 53, 54, 55, 83)
        THEN WDFArea
    END,
  PMFCArea =
    CASE
      WHEN WDFArea = 1
```

```
  THEN '5C'
WHEN WDFArea = 2
  THEN '5B'
WHEN WDFArea = 3
  THEN '5A'
WHEN WDFArea IN (4, 5, 6)
  THEN '3D'
WHEN WDFArea IN (7, 8, 9, 10, 11, 13)
  THEN '3C'
WHEN WDFArea IN (12, 14, 15)
  THEN '3B'
WHEN WDFArea IN (16, 17, 30)
  THEN '3A'
WHEN WDFArea IN (20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 83)
  THEN '4A'
WHEN WDFArea = 31
  THEN '5E'
WHEN WDFArea = 32
  THEN '2C'
WHEN WDFArea IN (33, 55)
  THEN '6A'
WHEN WDFArea = 34
  THEN '2B'
WHEN WDFArea = 36
  THEN '2A'
WHEN WDFArea = 38
  THEN '1C'
WHEN WDFArea = 40
  THEN '1B'
WHEN WDFArea = 42
  THEN '1A'
WHEN WDFArea = 48
  THEN '8A'
WHEN WDFArea = 49
  THEN '8B'
WHEN WDFArea = 50
  THEN '8C'
WHEN WDFArea = 51
  THEN '7C'
WHEN WDFArea = 52
```

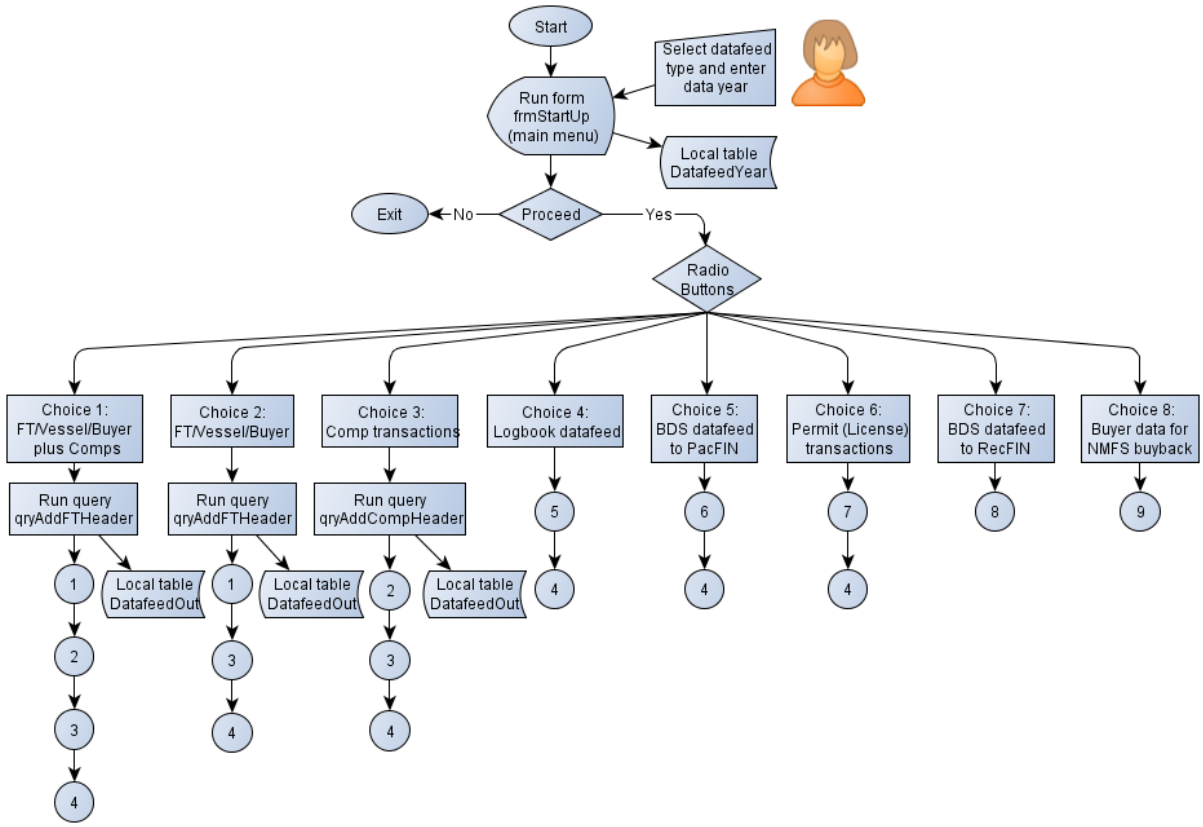
```
      THEN '7B'
    WHEN WDFArea = 53
      THEN '7A'
    WHEN WDFArea = 54
      THEN '6B'
    WHEN WDFArea = 99
      THEN '5D'
  END

SET NOCOUNT OFF
END

/* * ENABLE CHANGE TRACKING IN UPDATE TRIGGER */
UPDATE UTRIG SET TrackUpdates = 1

GRANT EXECUTE ON fill_missing_data TO dbo
```

# Appendix T: VBA Code Embedded in CTLS User Interface

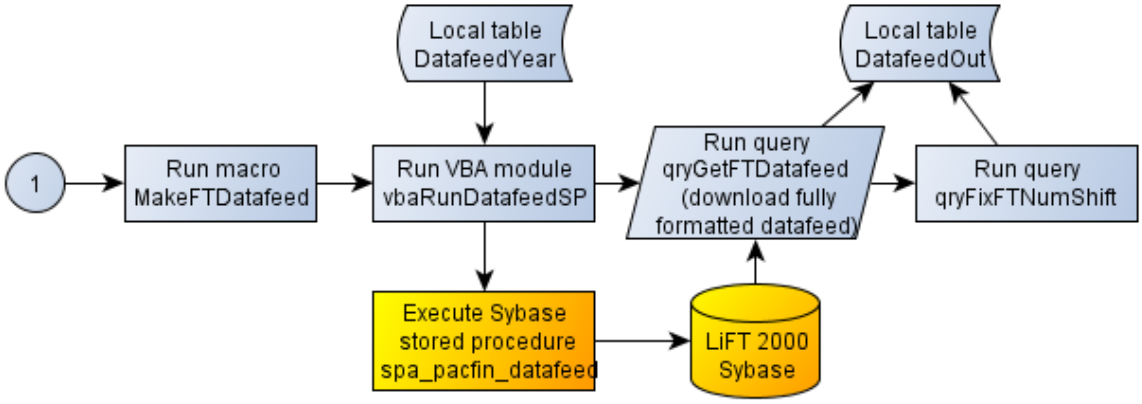
```
Private Sub Find_Log_Button_Click()
   FedDocNo.SetFocus
   DoCmd.FindRecord [FINDLOG Subform].Form!FedDocNo
   If FedDocNo = [FINDLOG Subform].Form!FedDocNo Then
      If DepartDate <> [FINDLOG Subform].Form!DepartDate Then
         DepartDate.SetFocus
         DoCmd.FindRecord [FINDLOG Subform].Form!DepartDate, , , , , , False
         FedDocNo.SetFocus
      End If
      If FedDocNo <> [FINDLOG Subform].Form!FedDocNo Or DepartDate
                  <> [FINDLOG Subform].Form!DepartDate Then
         DoCmd.FindRecord [FINDLOG Subform].Form!FedDocNo
         MsgBox ("Depart date not found")
      End If
   Else
      MsgBox ("Vessel not found")
   End If
End Sub

Private Sub Form_Load()
   FedDocNo.SetFocus
   DoCmd.FindRecord [FINDLOG Subform].Form!FedDocNo
   If DepartDate <> [FINDLOG Subform].Form!DepartDate Then
      DepartDate.SetFocus
      DoCmd.FindRecord [FINDLOG Subform].Form!DepartDate, , , , , , False
      FedDocNo.SetFocus
   End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
   [FINDLOG Subform].Form!FedDocNo.Value = FedDocNo.Value
   [FINDLOG Subform].Form!DepartDate.Value = DepartDate.Value
End Sub
```

# Appendix U: Process Flow Diagram for PacFIN/RecFIN Datafeed

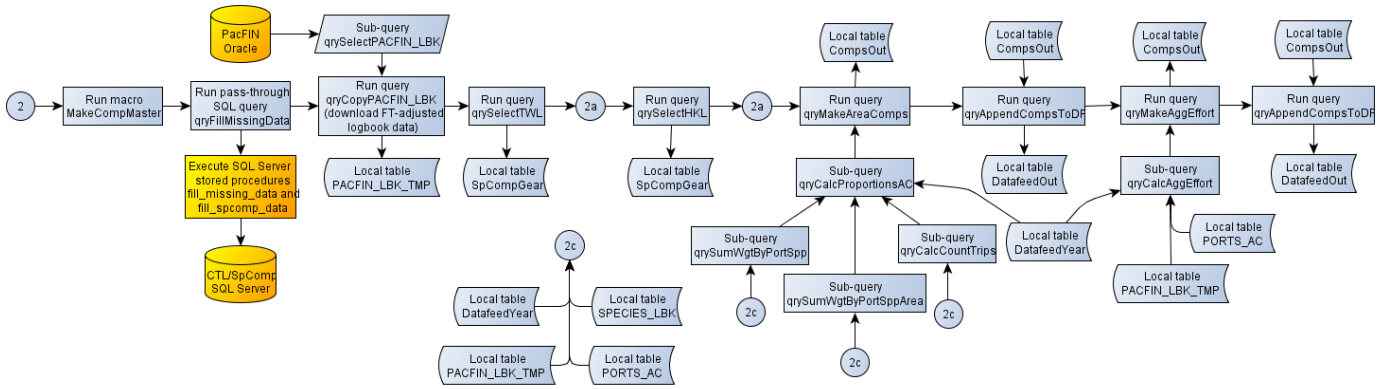
## Form Selection Menu



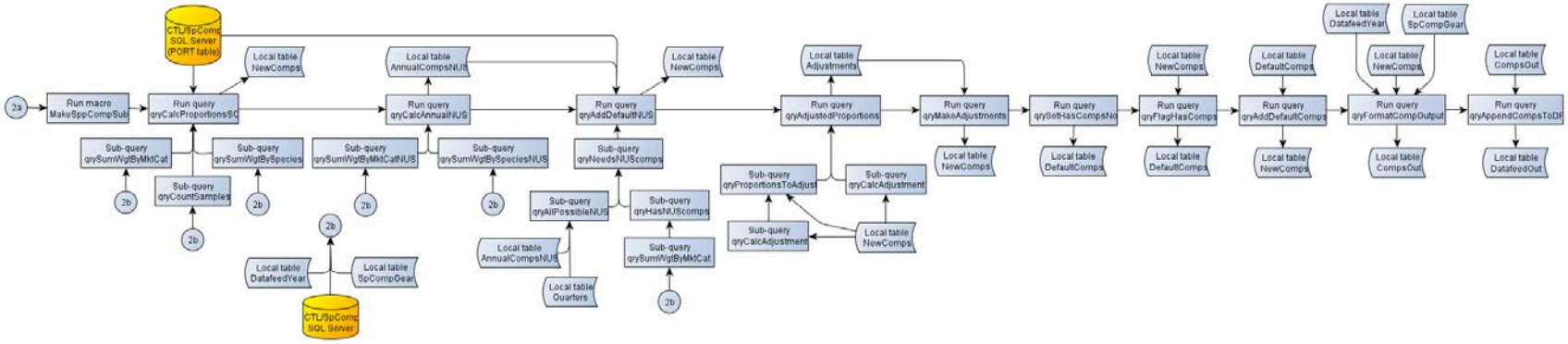
Sub-routine 1



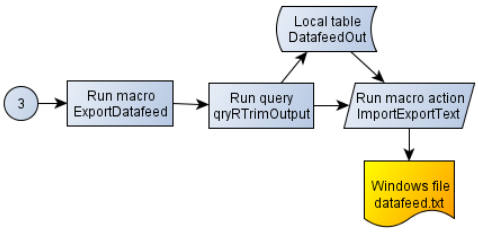
Sub-routine 2 and 2c



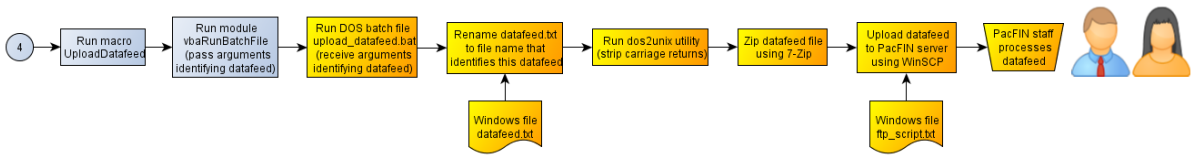
Sub-routine 2a and 2b



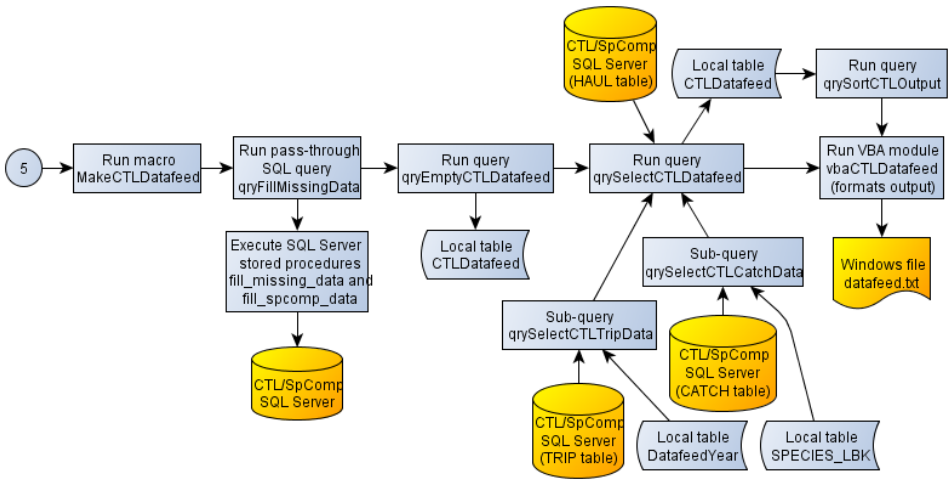
Sub-routine 3



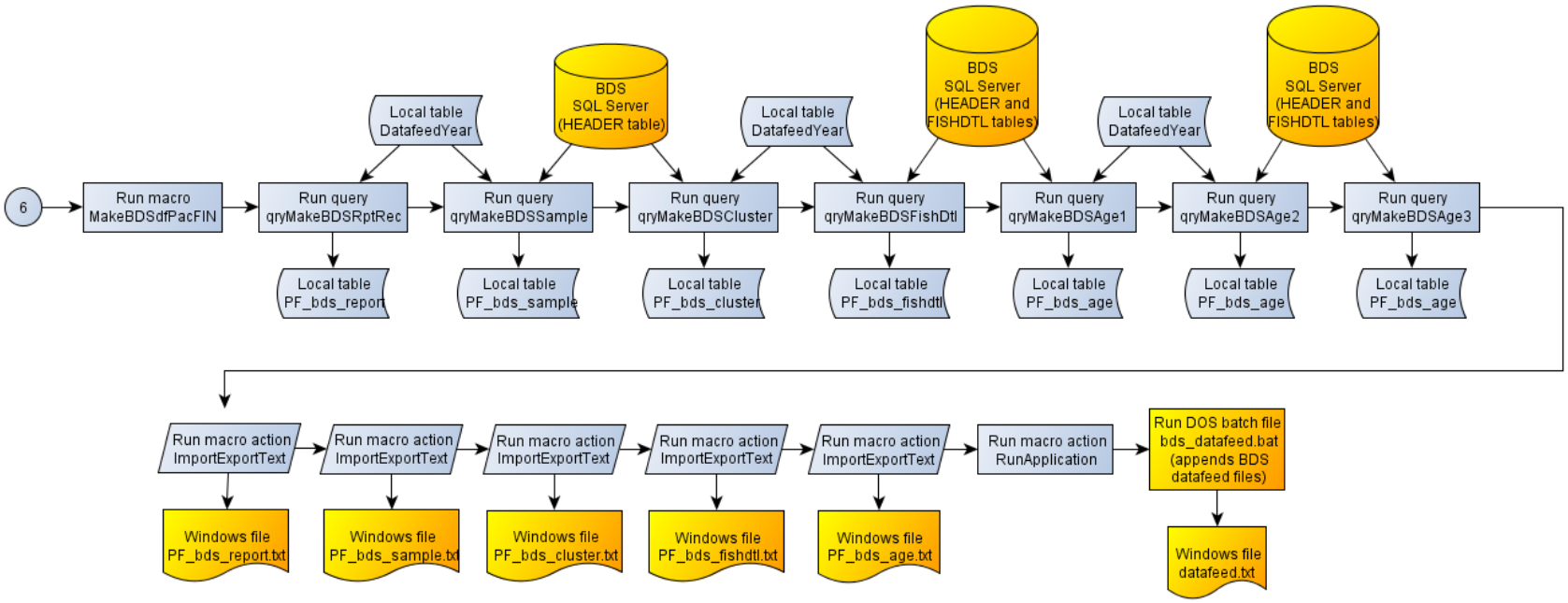

Sub-routine 4



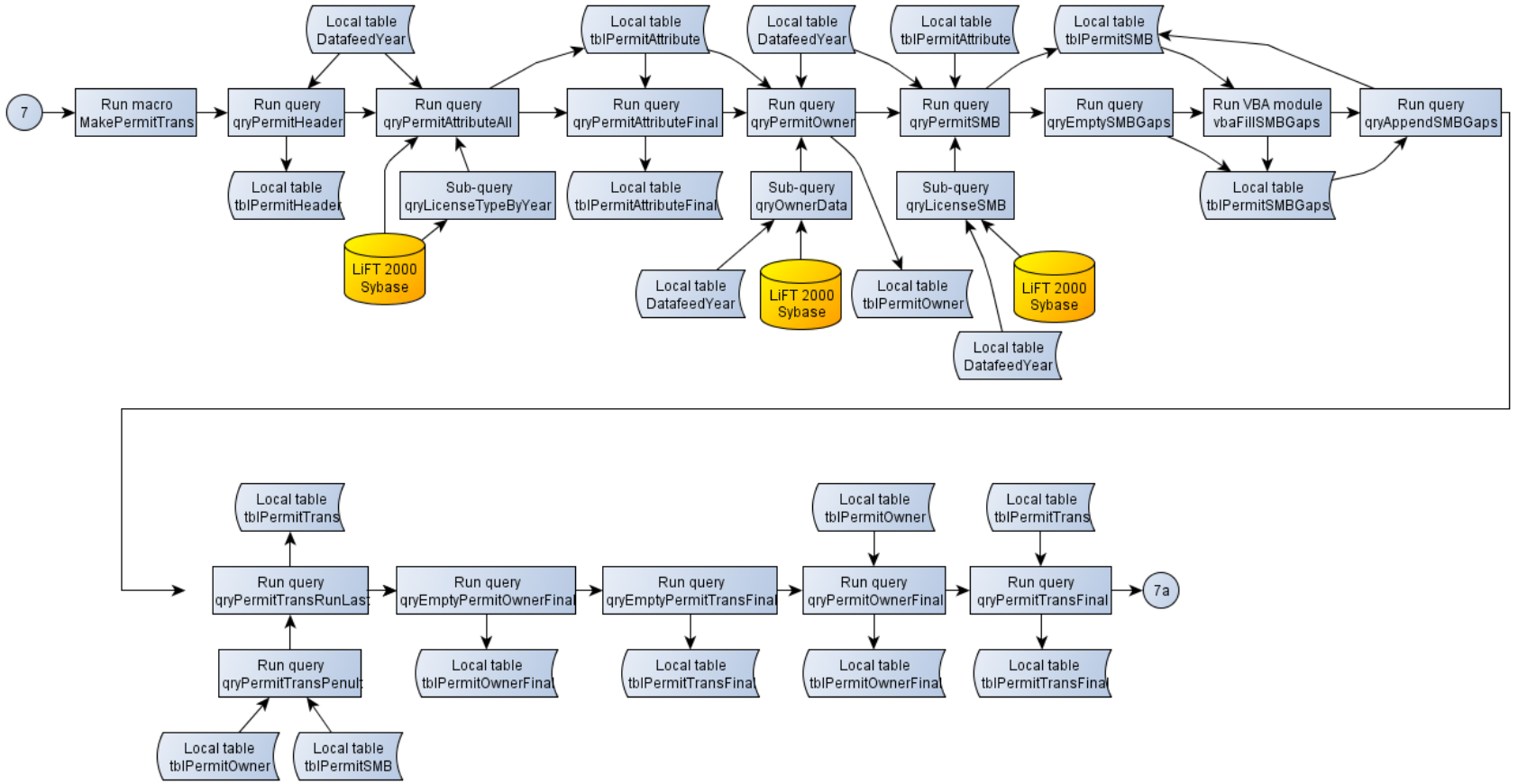

Sub-routine 5



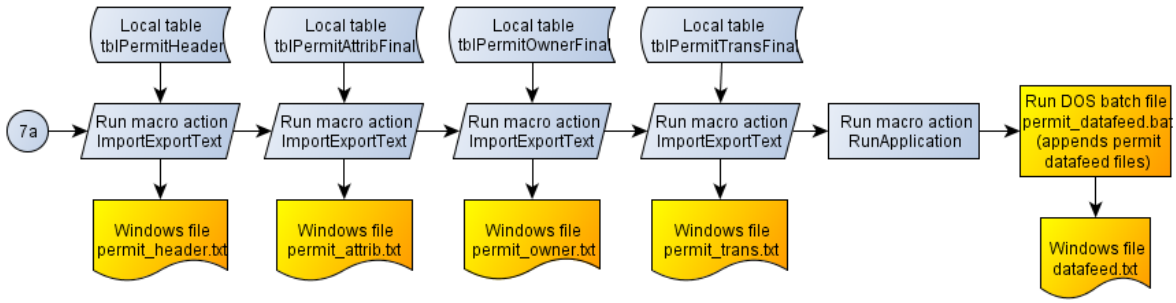
Sub-routine 6



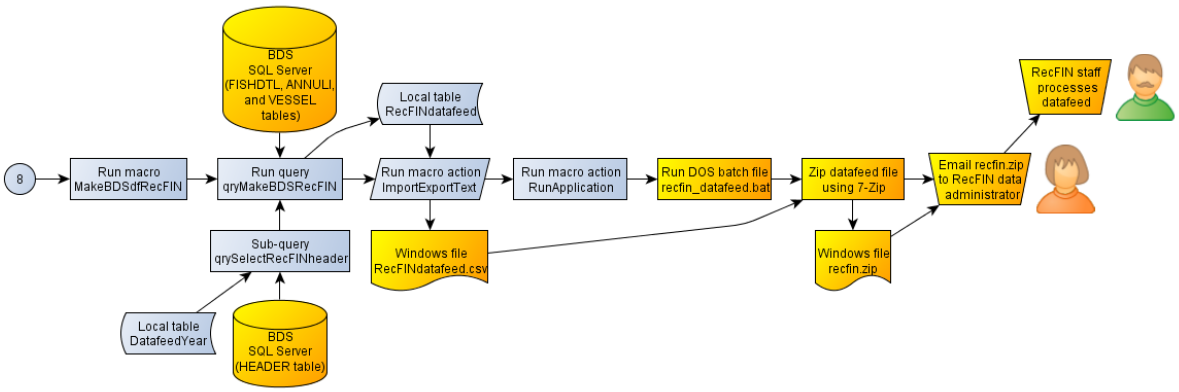
Sub-routine 7



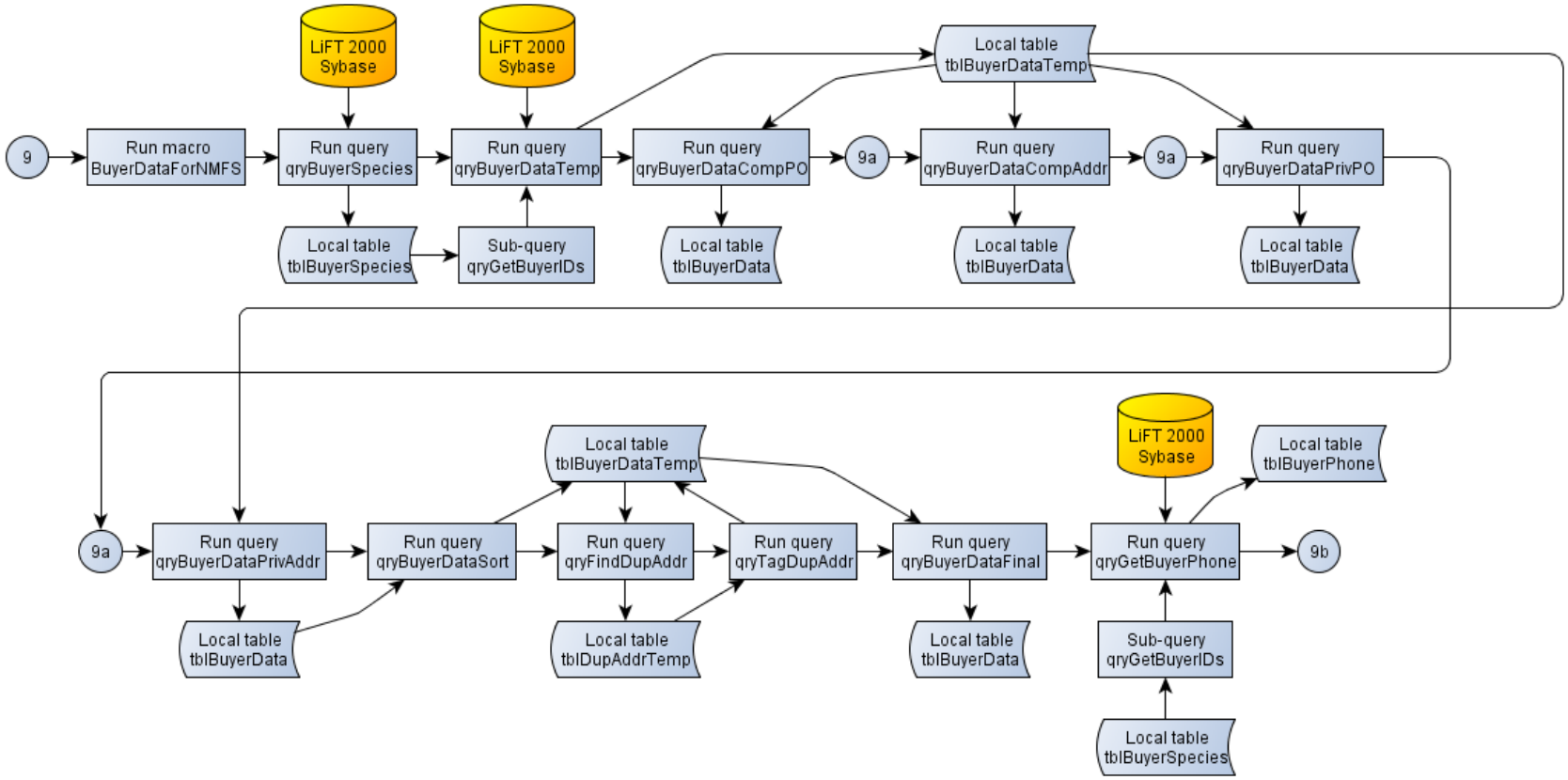
Sub-routine 7a



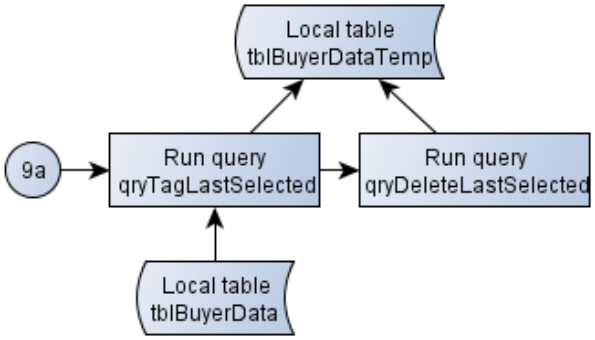

Sub-routine 8



Sub-routine 9



Sub-routine 9a




Sub-routine 9b