**Pesticides App**

**Internship project of:**

**Lorenzo Pozzi**

**(lorenzo.pozzi@mccain.ca | lorenzo.pozzi@wur.nl)**

**Application Functioning**

**&**

**Code Explanation**

**Table of Contents**

[Introduction 1](#_Toc203125895)

[Overview 1](#_Toc203125896)

[Quick Folders Descriptions 1](#_Toc203125897)

[Root folder 3](#_Toc203125898)

[Purpose 3](#_Toc203125899)

[Components 3](#_Toc203125900)

[main\_page folder 4](#_Toc203125901)

[Purpose 4](#_Toc203125902)

[Components 4](#_Toc203125903)

[Workflow 4](#_Toc203125904)

[Data Flow 5](#_Toc203125905)

[Key Features & Functionalities 5](#_Toc203125906)

[Integration with the Rest of the App 6](#_Toc203125907)

[user\_manual folder 7](#_Toc203125908)

[Purpose 7](#_Toc203125909)

[Components 7](#_Toc203125910)

[Workflow 7](#_Toc203125911)

[Data Flow 8](#_Toc203125912)

[Key Features & Functionalities 8](#_Toc203125913)

[Integration with the Rest of the App 9](#_Toc203125914)

[products\_page folder 10](#_Toc203125915)

[Purpose 10](#_Toc203125916)

[Components 10](#_Toc203125917)

[Workflow 11](#_Toc203125918)

[Data Flow 11](#_Toc203125919)

[Key Features & Functionalities 11](#_Toc203125920)

[Integration with the Rest of the App 12](#_Toc203125921)

[eiq\_calculator\_page folder 14](#_Toc203125922)

[Purpose 14](#_Toc203125923)

[Components 14](#_Toc203125924)

[Workflow 15](#_Toc203125925)

[Data Flow 15](#_Toc203125926)

[Key Features & Functionalities 16](#_Toc203125927)

[Integration with the Rest of the App 16](#_Toc203125928)

[season\_planner\_page folder 18](#_Toc203125929)

[Purpose 18](#_Toc203125930)

[Components 18](#_Toc203125931)

[Workflow 20](#_Toc203125932)

[Data Flow 21](#_Toc203125933)

[Key Features & Functionalities 21](#_Toc203125934)

[Integration with the Rest of the App 22](#_Toc203125935)

[data folder 23](#_Toc203125936)

[Purpose 23](#_Toc203125937)

[Components 23](#_Toc203125938)

[Workflow 24](#_Toc203125939)

[Data Flow 24](#_Toc203125940)

[Key Features & Functionalities 25](#_Toc203125941)

[Integration with the Rest of the App 26](#_Toc203125942)

[common folder 27](#_Toc203125943)

[Purpose 27](#_Toc203125944)

[Components 27](#_Toc203125945)

[Workflow 28](#_Toc203125946)

[Data Flow 29](#_Toc203125947)

[Key Features & Functionalities 29](#_Toc203125948)

[Integration with the Rest of the App 30](#_Toc203125949)

[Field use EIQ calculation logic 32](#_Toc203125950)

[Basic concepts and abbreviations 32](#_Toc203125951)

[Calculation process 32](#_Toc203125952)

[Data used and how it was produced 34](#_Toc203125953)

[UOM csv 34](#_Toc203125954)

[AI csv 35](#_Toc203125955)

[Products csv 36](#_Toc203125956)

Introduction

This is a desktop application that enables users (agronomists, field reps, and eventually growers) to view region-specific control products, compare them, and create crop protection plans, with a focus on EIQ scores, to encourage reduction of environmental impacts. The spray plans can be built from scratch or based on a previous year’s plan using the Excel format auto-generated by GXCore’s pesticides report, and re-exported in an Excel format that the app can re-read.

## Overview

Each folder in the codebase corresponds to a feature, and scripts are named by function (e.g., page\_abc.py for pages, tab\_abc.py for tabs within a page, widget\_abc.py for widgets, etc.). Within the code, I tried to maintain each function’s name as self explanatory as possible and wrote docstrings and comments explaining the behaviour of the more complex ones. The code and code documentation writing processes were aided mainly through the use of Anthropic AI’s Claude sonnet models 3.5, 3.7, and later 4 as they were published, but also with OpenAI’s Chat GPT-4o, and Google’s Gemini 2.0 flash for some minor tasks that Claude seemed to struggle with.

Before jumping into the code, I advise submitting to any AI chatbot the following prompts one by one, and reading [documentation](https://doc.qt.io/qtforpython-6/gettingstarted.html) as needed:

1. Concisely explain what PySide6 is and what is it used for.
2. In the context of PySide6: what are widgets? Concisely make some examples.
3. Concisely explain PySide6 signal-slot connection mechanism.
4. Concisely explain the Model–View–Controller architecture.

I also highly recommend, before contacting me, asking any AI chatbot for explanation of terms, structures, and features found throughout the code. I remain completely available for any question regarding operational and logic choices, or anything else you might want to ask me, my email addresses can be found in the title page.

## Quick Folders Descriptions

* **root folder**

Contains the main script (entry point), app icon, user\_preferences.json (persistent settings like row spacing and seeding rate), and the spec file for building the .exe.

* **main page**

Hosts scripts for the main window and home page, the widget for setting user preferences, and McCain and NA Ag logos.

* **user manual**

Contains a script that opens the manual in the default browser (avoids bundling HTML/CSS libraries for the .exe), the manual’s HTML, and all related screenshots.

* **products page**

Includes scripts for the products page, its two tabs (product list and comparison), and relevant widgets.

* **eiq calculator page**

Contains scripts for the EIQ calculator page, its tabs (single and multiple product calculators), and supporting widgets.

* **season planner page**

Scripts for scenario manager page, comparison page, scenario tabs, and subfolders:

* + import\_export: handles Excel import, parsing, and export
  + widgets: widgets used in these pages
  + models and delegates: implement the smart application table using the Model/View architecture of PySide6 (ask AI for a simple conceptual explanation if you are not familiar)
* **data**

Contains CSVs serving as local databases (to be removed when connecting to GXCore), and data models for:

* + AI (active ingredient)
  + Applications (single product use)
  + Products (control products)
  + Scenarios (spray plans with multiple applications)

Also includes AI, product, and UOM repositories that cache data at app launch for smooth performance. These repositories act as centralized sources for filtering and accessing objects data.

* **common**

Shared components, constants, and utility functions. Contains:

* + constants.py: app-wide constants (colors, EIQ thresholds, sizes, etc.)
  + styles.py: style functions and CSS
  + utils.py: utility functions (e.g., read/write preferences, EIQ coloring and rating, etc.)
  + widgets subfolder: shared widgets
  + **calculations** subfolder: core calculation engine, structured in 3 layers:
    - Interface (layer 1): public, user-friendly APIs used across the app
    - Layer 2: UOM standardizer (uses common/converter\_UOM.py and UOM repository)
    - Layer 3: simple math calculations using standardized units

**Note:** in some parts of this document and in the code, you will see references to a “tracer” to trace and debug calculations. This is a feature I initially introduced, that has been botched by agronomists feedback. I commented it out, but it is still in the code and can be reactivated. It could be useful for debugging and for the Farm assurance team in the future. This can easily be removed in the future if it is decided to do so.

# Root folder

## Purpose

Contains the main, the spec file, the file with the user preferences, and all the other folders that make the app function.

## Components

* **main.py**

serves as the application launcher. It initializes the Qt application, sets up repositories for products and active ingredients, loads configuration (user preferences), and creates the main window.

* **PesticidesApp.spec**

This is the PyInstaller specification file that packages the Python GUI application into a standalone Windows executable file. The spec file configures PySide6 as the GUI framework, includes openpyxl for Excel operations, and excludes numerous libraries including pandas, numpy, data science packages, unused Qt modules, and web frameworks to reduce the final executable size.

* **user\_preferences.json**

Contains the default values for user preferences, these can be modified within the app and saved, thus overwriting the default ones. This file enables persistency of the preferences between sessions.

* + default\_country: Canada
  + default\_region: New Brunswick
  + default\_row\_spacing: 34.0
  + default\_row\_spacing\_unit: inch
  + default\_seeding\_rate: 20.0
  + default\_seeding\_rate\_unit: cwt/acre

# main\_page folder

## Purpose

The main\_page module serves as the central hub for the application. It provides the main application window framework, home page, pages navigation, and user preference management. This module acts as the primary interface between users and the various features of the app.

## Components

**Core Files**

1. **window\_main.py** - Main application window container
   * MainWindow class: QMainWindow-based application container
   * Manages page navigation using QStackedWidget
   * Handles application-wide state and configuration
   * Has the yellow bar at the bottom with the user manual button and feedback links
2. **page\_home.py** - Basic home page implementation
   * HomePage class: Simple home page with navigation buttons
   * Lightweight version of the home interface
3. **widget\_preferences\_row.py** - User preferences management
   * PreferencesRow class: Horizontal widget for user settings
   * Manages country/region selection, row spacing, and seeding rate preferences

**Assets**

1. **logo\_McCain.png** - McCain Foods company logo
2. **logo\_NAAg.png** - North American Agriculture logo

## Workflow

**Application Initialization**

1. MainWindow initializes with configuration loading
2. Creates QStackedWidget for page management
3. Instantiates all application pages:
   * Home page (index 0)
   * Products page (index 1)
   * Scenarios manager (index 2)
   * EIQ calculator (index 3)
   * Scenarios comparison (index 4)
4. Sets up yellow status bar with user manual and feedback links

**User Navigation Flow**

Home Page → User selects feature → MainWindow navigates to appropriate page

↓

Preferences management → Filter changes → Notify all pages to refresh

↓

Page transitions → Check for unsaved preferences → Handle user confirmation

**Preference Management Workflow**

1. User modifies preferences in PreferencesRow
2. Changes marked as unsaved, Save button enabled
3. User saves → Configuration persisted → All pages notified
4. Navigation checks for unsaved changes → User prompted to save/discard

## Data Flow

**Signal-Slot Architecture**

PreferencesRow → HomePage → MainWindow → ProductRepository → All Pages

↓ ↓ ↓ ↓ ↓

User Input → Signal Relay → Filter Application → Data Update → UI Refresh

**Key Data Flows**

1. Preference Changes: PreferencesRow → HomePage → MainWindow → ProductRepository
2. Navigation: HomePage buttons → MainWindow.navigate\_to\_page()
3. Configuration: load\_config() → PreferencesRow → save\_config()

## Key Features & Functionalities

**Navigation System**

* Feature Buttons: Three main navigation buttons for core application features
  + Products List and Comparison (page 1)
  + EIQ Season Planner (page 2)
  + EIQ Calculator (page 3)

**Preference Management**

* Geographic Filtering: Country and region selection for product filtering
* Agricultural Settings: Row spacing and seeding rate configuration
* Persistence: Automatic saving and loading of user preferences
* Validation: Unsaved changes detection and user confirmation

**User Interface Elements**

* Dual Logo Header: McCain Foods and North American Agriculture branding
* Educational Content: EIQ explanation and Cornell University IPM links
* Status Bar: User manual access and feedback links
* Responsive Layout: ContentFrame-based design with consistent spacing

**State Management**

* Unsaved Changes Tracking: Prevents data loss during navigation
* Filter Coordination: Centralized product filtering across all pages
* Configuration Persistence: JSON-based user preferences storage

## Integration with the Rest of the App

**Repository Integration**

* Product Filtering: Applies geographic filters to ProductRepository
* Data Refresh: Coordinates product data updates across all pages

**Page Management**

* Stacked Widget: Manages transitions between major application sections
* Signal Coordination: Forwards preference changes to all relevant components

**External Dependencies**

* Common Widgets: Uses ContentFrame, SmartUOMSelector
* Styling System: Integrates with common.styles and common.constants
* Configuration: Interfaces with common.utils for settings management

**Cross-Module Communication**

MainWindow serves as the central hub for:

* Products Page: Refreshes product data when filters change
* Calculator Page: Updates available products for selection
* Season Planner: Synchronizes product lists and preferences
* User Manual: Provides help system integration

**Application Architecture Role**

The main page module acts as the orchestrator of the entire application, managing:

* Application lifecycle and window management
* User preference propagation
* Inter-page communication
* Resource loading and cleanup
* External service integration (feedback, user manual)

This module establishes the foundational framework that enables all other modules to function cohesively within the application.

# user\_manual folder

## Purpose

The user\_manual folder provides comprehensive documentation and help functionality for the application. It serves as the user assistance system, offering an interactive HTML-based manual that can be accessed directly from the application interface.

**User Journey Support**

The manual supports the complete user journey:

1. Onboarding: Initial setup and preferences configuration
2. Feature Discovery: Comprehensive feature documentation
3. Troubleshooting: Visual guides for common workflows
4. Advanced Usage: Complex operations like scenario import/export

This comprehensive manual ensures users can effectively utilize all features of the application while maintaining consistency with McCain's brand identity and user experience standards.

## Components

**Core Files**

1. user\_manual.html - The main HTML document containing the complete user manual with:
   * Modern, responsive web design using McCain's brand colors and guidelines
   * Interactive navigation sidebar
   * Comprehensive documentation for all app features
   * Screenshot galleries for visual guidance
   * Dismissible notification popup for user awareness that appearance may change according to screen size
2. service\_open\_user\_manual.py - Service module that:
   * Opens the user manual in the system's default web browser
   * Manages file path resolution and temporary directory creation
   * Manages asset copying for proper display
   * Provides error handling and user feedback

**Visual Assets**

The folder includes many screenshot files that provide visual guidance.

## Workflow

**Manual Access Process**

1. User Request: User clicks help button or menu item in the application
2. Service Invocation: service\_open\_user\_manual.py is called
3. Asset Preparation:
   * Creates temporary directory using tempfile.mkdtemp()
   * Copies entire manual directory to ensure all assets are available
   * Resolves file paths using common.utils.resource\_path
4. Browser Launch: Opens the manual in the system's default web browser
5. Error Handling: Shows appropriate error messages if any step fails

**Navigation Within Manual**

The HTML manual provides:

* Section-based Navigation: Home, Products, Calculator, Season Planner sections
* Interactive Sidebar: Click-to-navigate between sections
* Progressive Disclosure: Detailed feature explanations with visual aids

## Data Flow

Application Interface → Help Button Click → service\_open\_user\_manual.py

↓

Resource Path Resolution → Asset Copying → Temporary Directory Creation

↓

System Browser Launch → HTML Manual Display → User Interaction

## Key Features & Functionalities

**Design & User Experience**

* McCain Brand Consistency: Uses official McCain colors (#fee000 yellow, black, white, beige)
* Responsive Design: Adapts to different screen sizes
* Interactive Elements: Hover effects, smooth transitions, dismissible notifications
* Visual Hierarchy: Clear section organization with color-coded navigation

**Content Structure**

1. **Home Page Documentation**
   * Preferences configuration
   * Navigation guidance
   * EIQ information resources
2. **Products Page Documentation**
   * Product listing and filtering
   * Comparison functionality
   * Table operations and selection
3. **EIQ Calculator Documentation**
   * Single product calculator workflow
   * Multiple product comparison
   * Parameter configuration
4. **Season Planner Documentation**
   * Scenario management
   * Application table operations
   * Import/export procedures
   * Comparison views

**Technical Features**

* Asset Management: Automatic copying of CSS, images, and other resources
* Cross-Platform Compatibility: Works with system's default browser
* Error Handling: Graceful failure with user-friendly messages
* Temporary File Management: Clean temporary directory creation

## Integration with the Rest of the App

**Service Integration**

The manual service integrates with the main application through:

* Common Utilities: Uses common.utils.resource\_path for file path resolution
* Error Handling: Leverages PySide6's QMessageBox for consistent error reporting
* Help System: Accessible from various parts of the application interface

**Reference Points**

The manual is referenced in the main application structure:

* Build Configuration: Included in PesticidesApp.spec for PyInstaller packaging
* Help Buttons: Integrated into UI components across the application

# products\_page folder

## Purpose

The products\_page folder provides comprehensive product listing, filtering, and comparison functionality for the application. It serves as the primary interface for users to browse, search, and compare pesticide products, enabling informed decision-making through detailed product information and side-by-side comparisons.

## Components

**Main Page Component**

1. page\_products.py - Main container class that:
   * Manages the overall products page structure
   * Coordinates between listing and comparison tabs
   * Handles product data refresh and reset functionality
   * Provides navigation back to the home page

**Tab Components**

1. tab\_products\_list.py - Products listing tab that:
   * Displays searchable product table
   * Manages filtering interface
   * Handles product selection for comparison
   * Provides "Compare / View fact sheets" button
2. tab\_products\_comparison.py - Products comparison tab that:
   * Displays side-by-side product comparisons
   * Uses the ComparisonView widget for structured display
   * Shows detailed product properties and calculated EIQ values

**Widget Components**

1. widget\_products\_table.py - Table widget featuring:
   * ProductTable class with checkbox selection
   * NumericTableWidgetItem for proper numeric sorting
   * Predefined column structure with active ingredients and mode of action groups
   * Multi-column filtering capabilities
2. widget\_filter\_row.py - Modern filtering interface with:
   * FilterChip - Individual filter widgets with field selection
   * FiltersRow - Container managing up to 4 filter chips
   * Chip-style UI with field dropdowns and value inputs
3. widget\_comparison\_table.py - Comparison functionality with:
   * ComparisonTable - Side-by-side property comparison
   * Field EIQ calculation integration
   * Color-coded EIQ values with background highlighting
   * ComparisonView - Integrated comparison interface with table and EIQ

## Workflow

**Product Browsing Flow**

1. Page Load: ProductsPage initializes with tabs
2. Data Loading: ProductsListTab loads geographically filtered products from ProductRepository
3. Display: ProductTable renders products with predefined columns
4. Filtering: FiltersRow provides interactive filtering interface

**Product Selection and Comparison Flow**

1. Selection: Users check products in the table via ProductTable
2. Comparison Request: "Compare / View fact sheets" button triggers comparison
3. Tab Switch: System navigates to comparison tab automatically
4. Comparison Display: ComparisonView renders side-by-side comparison
5. EIQ Calculation: Field EIQ values calculated using eiq\_calculator

**Reset and Refresh Flow**

1. Reset Trigger: User clicks Reset button in header
2. Data Refresh: ProductRepository.refresh\_from\_csv() reloads data
3. UI Reset: All selections cleared, filters reset, comparison cleared
4. Tab Navigation: Returns to products list tab

## Data Flow

CSV Data → ProductRepository → ProductsListTab → ProductTable → User Selection

↓

FilterChip Selection → FiltersRow → ProductTable.apply\_filters()

↓

Selected Products → ComparisonTab → ComparisonTable → EIQ Calculator → Display

**Key Data Transformations**

* Product Objects: Raw CSV data → Product objects → Table display
* Filter Criteria: User input → (column\_index, filter\_text) tuples → Row visibility
* Comparison Data: Selected products → Property dictionary → Side-by-side table
* EIQ Calculation: Product data → eiq\_calculator → Color-coded display

## Key Features & Functionalities

**Advanced Table Features**

* Checkbox Selection: Multi-select products with visual feedback
* Numeric Sorting: Proper sorting for REI and PHI columns using NumericTableWidgetItem
* Column Configuration: Predefined column structure with fixed and stretch widths
* Active Ingredient Display: Consolidated AI groups with organization formatting

**Filtering System**

* Chip-Based UI: Filter “chips” with field selection and value input
* Multi-Filter Support: Up to 4 simultaneous filters
* Dynamic UI: Add/remove filters with visual feedback
* Real-time Filtering: Instant table updates as filters change

**Comprehensive Comparison**

* Side-by-Side Display: Properties as rows, products as columns
* Field EIQ Integration: Calculated EIQ values with color coding
* Predefined Properties: Curated list of important product attributes
* Visual Hierarchy: Color-coded EIQ values and bold formatting for key fields

**Data Management**

* Repository Integration: Seamless data loading from ProductRepository
* Reset Functionality: Complete data refresh and UI reset
* Error Handling: Graceful handling of missing data and calculation errors

## Integration with the Rest of the App

**Core System Integration**

* Navigation: Integrates with main application navigation via HeaderWithHomeButton
* Styling: Uses consistent styling from common.styles
* Repository: Connects to ProductRepository for data access
* Configuration: Uses get\_config for user preferences

**Calculation Integration**

* EIQ Calculator: Direct integration with eiq\_calculator
* Calculation Tracing: Uses calculation\_tracer for debugging
* Color Coding: Leverages get\_eiq\_color for EIQ value visualization

**UI Component Integration**

* Content Frames: Uses ContentFrame for consistent layouts
* Button Styling: Uses with create\_button system from common.widgets.header\_frame\_buttons.py
* Font Management: Uses common font sizing and styling functions from common.styles.py

**Data Model Integration**

* Product Model: Works with Product objects and their methods
* Dictionary Conversion: Uses product.to\_dict() for table display
* AI Data Access: Leverages product.get\_ai\_data() for calculations

**Application Architecture**

The products page serves as a central hub connecting:

* Data Layer: Product repository and models
* Calculation Layer: EIQ calculation system
* UI Layer: Modern filtering and comparison interfaces
* Navigation Layer: Main application flow

This comprehensive integration ensures the products page provides a seamless user experience while maintaining consistency with the overall application architecture, UX, and McCain's brand standards.

# eiq\_calculator\_page folder

## Purpose

The eiq\_calculator\_page folder provides comprehensive EIQ calculation UI functionality for the application. Note that it provides the User Interface, but not the calculation logic itself. It enables users to calculate and compare the environmental impact of pesticide applications through both single product calculations and multi-product comparisons, supporting informed decision-making for sustainable pest management.

## Components

**Main Page Component**

1. page\_eiq\_calculator.py - Main container class that:
   * Manages the overall EIQ calculator page structure
   * Coordinates between single and comparison calculator tabs
   * Provides reset functionality for both calculators
   * Handles navigation back to the home page

**Tab Components**

1. tab\_single\_calculator.py - Single product calculator featuring:
   * SingleProductCalculatorTab class for individual product EIQ calculations
   * Product selection and active ingredients display
   * Application parameters configuration
   * Real-time EIQ calculation and results display using the color-bar
2. tab\_multi\_calculator.py - Multi-product comparison calculator with:
   * ProductComparisonCalculatorTab class for comparing multiple products
   * Intuitive card-based interface for product selection
   * Side-by-side EIQ comparison table
   * Horizontal scrolling for multiple product cards

**Widget Components**

1. widget\_product\_card.py - Product card widget featuring:
   * ProductCard class for individual product input
   * Product selection with active ingredients display
   * Application parameters configuration
   * Remove functionality with signal emissions
2. widgets\_results\_display.py - Results display widgets including:
   * EiqResultDisplay - Single product results with score bar
   * EiqComparisonTable - Multi-product comparison table
   * ColorCodedEiqItem - Color-coded EIQ values in tables

## Workflow

**Single Product Calculator Flow**

1. Product Selection: User selects product from ProductSelectionWidget
2. Data Loading: System loads product data from ProductRepository
3. Information Display:
   * Active ingredients table populated with AI data
   * Label information table shows product specifications
   * Application parameters pre-filled with product defaults
4. Parameter Configuration: User adjusts rate, unit, and applications
5. EIQ Calculation: Real-time calculation using eiq\_calculator
6. Results Display: EiqResultDisplay shows Field EIQ with score bar

**Multi-Product Comparison Flow**

1. Card Management: Users add/remove ProductCard widgets dynamically
2. Product Selection: Each card allows independent product selection
3. Parameter Configuration: Individual application parameters per card
4. EIQ Calculation: Parallel calculations for all configured cards
5. Comparison Display: EiqComparisonTable shows side-by-side results
6. Dynamic Updates: Table updates automatically when card data changes

**Reset and Data Refresh Flow**

1. Reset Trigger: User clicks Reset button in header
2. Data Refresh: Both tabs refresh product data from repository
3. UI Reset: All selections cleared, parameters reset, results cleared
4. Tab Navigation: Returns to single product calculator tab

## Data Flow

**Single Calculator Data Flow**

ProductRepository → Product Selection → Active Ingredients Data → Display Tables

↓

Application Parameters → EIQ Calculator → User Preferences → Field EIQ Result

↓

EiqResultDisplay → ScoreBar → Color-coded Visual Feedback

**Multi Calculator Data Flow**

ProductCard Array → Individual Product Data → Parallel EIQ Calculations

↓

Card Index Management → Dynamic Table Updates → EiqComparisonTable

↓

Color-coded Results → Product Comparison → Decision Support

**Key Data Transformations**

* Product Objects: CSV data → Product objects → UI display
* Active Ingredients: Product data → get\_ai\_data() → Table display
* EIQ Calculation: AI data + parameters → calculate\_product\_field\_eiq() → Field EIQ
* Color Coding: EIQ values → get\_eiq\_color() → Visual representation

## Key Features & Functionalities

**Single Product Calculator Features**

* **Comprehensive Product Display**:
  + Active ingredients table with concentrations and UOM
  + Label information table with rates, REI, PHI, and application intervals
  + Pre-populated application parameters based on product label data
* **Real-time Calculation**:
  + Automatic EIQ calculation on parameter changes
  + Integration with calculation\_tracer for debugging
  + Score bar visualization with impact ratings
* **Smart Parameter Management**:
  + Application rate is auto converted when UOM is changed
  + Default values from product label information

**Multi-Product Comparison Features**

* **Dynamic Card Interface**:
  + Add/remove product cards dynamically
  + Horizontal scrolling for multiple products
  + Individual parameter configuration per product
* **Intelligent Comparison**:
  + Side-by-side EIQ comparison table
  + Color-coded results with ColorCodedEiqItem
  + Automatic table updates on data changes
* **Card Management**:
  + Unique product identification with card indexing
  + Automatic title updates when cards are removed
  + Signal-based communication for data changes

## Integration with the Rest of the App

**Core System Integration**

* Navigation: Integrates with main application navigation via HeaderWithHomeButton
* Data Repository: Connects to ProductRepository for filtered product access
* Configuration: Uses get\_config for user preferences in calculations

**Calculation System Integration**

* EIQ Engine: Direct integration with eiq\_calculator.calculate\_product\_field\_eiq()
* Tracing: Uses calculation\_tracer for detailed calculation logging
* User Preferences: Incorporates user-defined UOM conversion preferences

**UI Component Integration**

* Common Widgets: Leverages several widgets from the common.widgets folder
* Styling: Uses consistent styling from common.styles
* Content Frames: Utilizes ContentFrame for consistent layouts
* Score Bar: Integrates ScoreBar with calculator preset configuration
* Color Coding: Uses get\_eiq\_color and get\_eiq\_rating for consistent visual feedback
* Font Management: Leverages get\_subtitle\_font for consistent typography

**Data Model Integration**

* Product Data: Works with Product objects and their methods
* Active Ingredients: Uses product.get\_ai\_data() for calculation input
* Constants: Integrates with EIQ\_\*\_THRESHOLD values for impact classification

**Application Architecture**

The EIQ calculator page serves as a central computational hub connecting:

* Data Layer: Product repository and models
* Calculation Layer: EIQ calculation engine with tracing
* UI Layer: Modern tabbed interface with dynamic components
* Configuration Layer: User preferences and application settings

This comprehensive integration ensures the EIQ calculator provides accurate, traceable calculations while maintaining consistency with the overall application architecture and McCain's user experience standards. The dual-calculator approach supports both single-product assessments and multi-product comparisons for comprehensive pest management decision-making.

# season\_planner\_page folder

## Purpose

The season\_planner\_page provides comprehensive seasonal pesticide application planning and management functionality for the application. It enables users to create, manage, and compare seasonal crop protection plans (referred to as “scenarios”) through an Excel-like interface, supporting multi-scenario planning with detailed EIQ impact analysis and data import/export capabilities using GXCore generated reports.

Note: the applications table uses the MVC architecture, find more information here.

## Components

**Main Page Components**

1. page\_scenarios\_manager.py - Main scenarios management page featuring:
   * ScenariosManagerPage class for overall scenario coordination
   * Tab interface for multiple scenarios with custom tab bar
   * Action buttons for scenario operations (New, Clone, Delete, Import, Export, etc.)
   * Real-time EIQ summary display with regenerative agriculture framework classes
2. page\_sceanrios\_comparison.py - Scenarios comparison page with:
   * ScenariosComparisonPage class for side-by-side scenario analysis
   * Horizontal scrolling layout for multiple scenario tables
   * Visual comparison of EIQ impacts across scenarios and applications

**Tab Component**

1. tab\_scenario.py - Individual scenario tab containing:
   * ScenarioTabPage class for single scenario management
   * Metadata widget for scenario information
   * Smart applications table for pesticide application details
   * Add and remove applications buttons

**Widget Components**

1. widgets/metadata\_row.py - Scenario metadata widget:
   * SeasonPlanMetadataWidget for crop year, grower, and field information
   * Field area input with SmartUOMSelector for area units
   * Real-time metadata change tracking
2. widgets/scenario\_comparison\_table.py - Comparison display widget:
   * ScenarioComparisonTable for scenario summaries
   * Product type grouping with EIQ values
   * Applications sorting by EIQ contribution
   * Color-coded EIQ values and regenerative agriculture classification
3. widgets/eiq\_summary.py - EIQ summary widget:
   * EIQSummaryWidget with score bar visualization
   * Regenerative agriculture framework thresholds and classifications
   * Application count and total EIQ display
4. widgets/applications\_table.py - Core applications smart table widget featuring:

* ApplicationsTableWidget class for Excel-like applications management
* Advanced Table Interface: QTableView with specialized delegates for each column type
* Delegate Management: Comprehensive delegate system including:
  + ReorderDelegate with move up/down functionality
  + DateDelegate for application date selection
  + ProductNameDelegate with product dropdown
  + UOMDelegate for application rate units
  + RateDelegate and AreaDelegate for numeric inputs
  + MethodDelegate for application methods
  + ProductTypeDelegate for product categorization
* Real-time Data Management: Direct integration with ApplicationTableModel
* Application Operations: Add, remove, and reorder applications with user confirmation
* Signal Integration: Emits eiq\_changed and applications\_changed signals for real-time updates
* Selection Management: Single-row selection with visual feedback and keyboard navigation (up and down)
* Data Validation: Integration with validation system through model layer
* Product Data Refresh: Capability to refresh product data when repository changes

**Model Components**

1. models/application\_table\_model.py - Table data model:
   * ApplicationTableModel extending a base QAbstractTableModel
   * Cell data management, background colors, and tooltips handling
   * Integration with validation and EIQ calculation systems
2. models/applications\_eiq\_calculator.py - EIQ calculation model:
   * Real-time EIQ calculations for individual applications
   * Handling of products with missing EIQ value from Cornell
   * Real-time EIQ calculations for overall scenario
3. models/application\_validator.py - Data validation model:
   * Application data validation with ValidationState tracking
   * Input validation for rates, dates, and product selections

**Delegate Components**

1. **delegates/** folder containing specialized cell editors:
   * date\_delegate.py - Date picker for application dates (generic string, not necessarily a date)
   * product\_name\_delegate.py - Product selection dropdown
   * product\_type\_delegate.py – Product type selection dropdown
   * uom\_delegate.py - UOM selection for rates (SmartUOMSelector)
   * numeric\_delegate.py - Numeric input validation (used for area and rate)
   * method\_delegate.py - Application method selection

**Import/Export Components**

1. import\_export/import\_dialog.py - Excel import functionality:
   * ImportScenarioDialog for file selection and data mapping
   * ProductMappingWidget for handling unmatched products
2. import\_export/exporter.py - Excel export functionality:
   * ExcelScenarioExporter for multi-scenario export
   * ScenarioDataWriter for structured worksheet creation
3. import\_export/excel\_parser.py - Excel parsing utilities:
   * Data extraction and validation from Excel files
   * Product matching and data transformation

## Workflow

**Scenario Management Flow**

1. Scenario Creation: User creates new scenario via "New Scenario" button (or importing one, see below)
2. Metadata Entry: SeasonPlanMetadataWidget captures crop year, grower, field details
3. Application Planning: Excel-like table interface for entering pesticide applications
4. Real-time Validation: ApplicationValidator ensures data completeness
5. EIQ Calculation: Automatic Field EIQ computation using ApplicationEIQCalculator
6. Results Display: EIQSummaryWidget shows total impact within regenerative agriculture classification

**Comparison Workflow**

1. Multi-Scenario Selection: User creates or loads multiple scenarios
2. Comparison Request: "Compare Scenarios" button launches comparison page
3. Side-by-Side Display: scenariosComparisonPage shows scenarios side to side
4. Product Grouping: ScenarioComparisonTable groups applications by product type and orders and colors them by EIQ contribution
5. EIQ Analysis: Visual comparison of Field EIQ values across scenarios to help reduce scores

**Import/Export Workflow**

1. Import Process:
   * File selection via ImportScenarioDialog
   * Excel parsing with excel\_parser
   * Unmatched product management (skip, import as-is, map to existing product) using ProductMappingWidget
   * Scenario creation and tab addition
2. Export Process:
   * Multi-scenario selection for export
   * ExcelScenarioExporter creates structured Excel file, one scenario per worksheet
   * Metadata and application data export with formatting

## Data Flow

**Core Data Flow**

Scenario Object ↔ ScenarioTabPage ↔ ApplicationTableModel ↔ QTableView & Delegates

                                    ↓

Application Objects → ApplicationEIQCalculator → Field EIQ Values

                                    ↓

EIQ Results → EIQSummaryWidget → Score Bar Display → Regen Agriculture Classification

**Import Data Flow**

Excel File → ExcelParser → Product Matching → Application Objects creation → Scenario Creation → Tab Addition

**Export Data Flow**

Scenario Objects → ScenarioDataWriter → Excel Worksheets → File Export

**Key Data Transformations**

* Scenario Data: Scenario objects ↔ UI components ↔ Excel format
* Application Data: Application objects ↔ Table cells ↔ EIQ calculations
* Field Area: Metadata widget → Application calculator → EIQ computation
* Product Matching: Excel product names → Database products → Application validation

## Key Features & Functionalities

**Advanced Table Management**

* Excel-like Interface: QTableView with custom delegates for specialized cell editing
* Real-time Validation: ApplicationValidator with visual feedback and smart tips
* Smart Delegates: Context-aware editors for dates, products, UOMs, and numeric values

**Comprehensive EIQ Analysis**

* Real-time Calculation: Automatic Field EIQ computation on data changes
* Regenerative Agriculture Framework: Classification using thresholds (Leading, Advanced, Engaged, Onboarding)
* Visual Feedback: ScoreBar with color-coded impact levels
* Product Type Grouping: Organized analysis by pesticide categories

**Multi-Scenario Management**

* Tabbed Interface: Custom tab bar with drag-to-reorder functionality
* Scenario Operations: Clone, delete, import, export scenarios
* Name Management: Unique name generation and rename functionality
* State Tracking: Real-time updates of scenario status and validation

**Import/Export Capabilities**

* Excel Integration: Full import/export support for scenario data (limited to GXCore report format and app generated file format)
* Product Mapping: Intelligent matching of Excel products to database
* Data Validation: Import validation with user feedback
* Structured Export: Multi-scenario Excel files with metadata and formatting

**Advanced Comparison Features**

* Side-by-Side Analysis: Horizontal scrolling comparison layout
* Product Type Grouping: Applications grouped by pesticide category with EIQ sorting
* Visual Differentiation: Color-coded EIQ values and framework classifications
* Summary Statistics: Total EIQ and application counts per scenario

## Integration with the Rest of the App

**Core System Integration**

* Navigation: Integrates with main application navigation via HeaderWithHomeButton
* Data Repository: Connects to ProductRepository for product data access
* Configuration: Uses get\_config for user preferences and settings

**Data Model Integration**

* Scenario Management: Works with Scenario and Application objects
* Product Integration: Leverages Product objects for application validation
* Repository Pattern: Uses repository pattern for data retrieval

**Calculation System Integration**

* EIQ Engine: Direct integration with eiq\_calculator for Field EIQ computation
* Area Conversion: Uses UOM conversion system
* Regenerative Agriculture: Integrates get\_regen\_ag\_class for framework classification

**UI Component Integration**

* Common Widgets: Leverages SmartUOMSelector, ScoreBar, and ContentFrame among the other
* Styling System: Uses consistent styling from common.styles for several objects
* Font Management: Integrates with font system for consistent typography

**Import/Export Integration**

* File Dialogs: Uses Qt file dialogs for Excel file selection
* Error Handling: Consistent error messaging using QMessageBox

**Application Architecture**

The season planner serves as a comprehensive planning hub connecting:

* Data Layer: Scenario and application models with repository access
* Calculation Layer: EIQ calculation engine with regenerative agriculture framework
* UI Layer: Advanced table interface with specialized delegates
* Import/Export Layer: Excel integration for external data exchange
* Comparison Layer: Multi-scenario analysis and visualization

This comprehensive integration ensures the season planner provides a complete pest management planning solution while maintaining consistency with the overall application architecture and McCain's sustainability goals through regenerative agriculture framework integration.

# data folder

## Purpose

The data folder serves as the core data access layer for the application, providing centralized data management for pesticide products, active ingredients, units of measurement, and agricultural scenarios. It implements a repository pattern with model classes to ensure consistent data access, validation, and transformation throughout the application.

## Components

**Model Classes**

1. model\_product.py - Product data model:
   * Product class representing individual pesticide products
   * Product properties, active ingredients, and label information
   * EIQ calculation integration and data transformation methods
2. model\_AI.py - Active ingredient model:
   * ActiveIngredient class for pesticide active ingredients
   * Mode of action groups and classification data
   * EIQ values
3. model\_application.py - Application data model:
   * Application class representing individual pesticide (Product class) applications
   * Application parameters, timing, and method information
   * Integration with scenario planning and EIQ calculations
4. model\_scenario.py - Scenario data model:
   * Scenario class for seasonal pest management plans
   * Metadata management (crop year, grower, field information)
   * Application collection and scenario-level operations

**Repository Classes**

1. repository\_product.py - Product data repository:
   * ProductRepository class for centralized product data access
   * CSV loading, caching, and filtering capabilities
   * Product search and selection functionality
2. repository\_AI.py - Active ingredient repository:
   * AIRepository class for active ingredient data management
   * CSV loading and caching
   * Mode of action group lookup and classification
   * EIQ data access for calculations
3. repository\_UOM.py - Unit of measurement repository:
   * UOMRepository class for UOM paring, conversion, and management
     1. BaseUnit class for fundamental UOMs
     2. CompositeUOM class for composite unit X/Y (like kg/ha, etc.)
   * UOM category organization and conversion factor access
   * Integration with measurement and calculation systems

**Data Conversion Utilities**

1. converter\_UOM.py - Unit conversion engine:
   * Support unit for the UOMRepository class with all the complex formulas
   * Unit conversion calculations and validation
   * Support for multiple measurement categories (area, volume, weight)
   * Integration with user preferences and application parameters

**Data Files**

1. CSV Data Sources (more on them in their following chapter):
   * csv\_products.csv - Pesticide products database
   * csv\_AI.csv - Active ingredients data
   * csv\_UOM.csv - Unit of measurement definitions and conversion factors

## Workflow

**Data Loading and Initialization Flow**

1. Application Startup: Repositories initialize and load CSV data using resource\_path
2. Data Caching: Repository classes cache loaded data for performance optimization
3. Model Creation: CSV data transformed into model objects (Product, ActiveIngredient, etc.)
4. Data Validation: Model classes validate data integrity and relationships
5. Repository Registration: Repositories become available for application-wide access

**Product Data Access Flow**

1. Repository Query: UI components request product data from ProductRepository
2. Filtering: Repository applies search criteria and filters
3. Data Enrichment: Products linked with active ingredient data from AIRepository
4. UI Display: Enriched product objects displayed in UI components

**Scenario Management Flow**

1. Scenario Creation: New Scenario objects created with metadata
2. Application Addition: Application objects added to scenarios
3. Data Validation: Applications validated against product and AI repositories
4. EIQ Calculation: Applications integrated with calculation engine for impact assessment
5. Persistence: Scenarios can be exported/imported via Excel integration

## Data Flow

**Core Data Architecture**

CSV Files → Repository Classes → Model Objects → UI Components → User Interaction

    ↓               ↓                  ↓               ↓

Caching → Filtering → Validation → Display → User Input

    ↓               ↓                  ↓               ↓

Performance → Search → Integrity → Interaction → Data Updates

**EIQ Calculation Data Flow**

Product Data → Active Ingredient Data → EIQ Components → Calculation Engine

                        ↓                       ↓                ↓

Application Parameters → UOM Conversion → Field EIQ Results → UI Display

**Repository Pattern Data Flow**

UI Request → Repository Query → CSV Data Access → Model Creation → Data Return

               ↓                     ↓        ↓               ↓

             Caching → File Reading → Object Creation → Validation → Response

**Key Data Transformations**

* CSV to Objects: Raw CSV data → Repository caching → Model object creation
* Product Enrichment: Product records + AI data → Complete product information
* Unit Conversion: User input + UOM data → Standardized measurements
* EIQ Integration: Product/AI data → EIQ components → Environmental impact scores

## Key Features & Functionalities

**Repository Pattern Implementation**

* Centralized Data Access: Single point of access for each data type, having one unique source of truth for simplicity and coherency
* Caching Strategy: In-memory caching for performance optimization
* Data Refresh: Capability to reload data without application restart

**Model Object Features**

* Data Validation: Built-in validation for data integrity and relationships
* Business Logic: Model methods for calculations and data transformations
* Serialization: Support for data export/import through various formats
* Relationship Management: Proper linking between products, AIs, and applications

**Advanced Filtering and Search**

* Multi-criteria Filtering: Complex filtering across multiple product attributes
* Text Search: Case-insensitive search across product names and descriptions
* Category Filtering: Filtering by product type, mode of action, and other classifications

**Unit Conversion System**

* Multi-category Support: Area, volume, weight, and concentration conversions
* Conversion Factors: Accurate conversion between different measurement units
* User Preferences: Integration with user-defined unit preferences for conversion of seed treatment rates and in-furrow products (expressed as e.g., l/100m)
* Validation: Input validation and error handling for unit conversions

**Data Integrity Features**

* Referential Integrity: Validation of relationships between data entities
* Data Consistency: Ensuring data consistency across related objects
* Error Handling: Graceful handling of missing or invalid data
* Data Validation: Comprehensive validation rules for all data types

## Integration with the Rest of the App

**Core Application Integration**

* Repository Access: All application components access data through repository pattern
* Model Usage: UI components work directly with model objects for type safety
* Calculation Integration: Direct integration with eiq\_calculator system
* Configuration: Integration with get\_config for user preferences

**UI Component Integration**

* Product Selection: ProductSelectionWidget uses ProductRepository
* Application Tables: Season planner uses Application and Scenario models
* Comparison Views: Product comparison features leverage repository filtering
* EIQ Displays: EIQ widgets integrate with Product and AI model data for calculations

**Calculation System Integration**

* EIQ Engine: Model objects provide data for eiq\_calculator
* Active Ingredient Data: AIRepository supplies EIQ component values
* Unit Conversion: converter\_UOM integrates with calculation parameters
* Field Calculations: Application models provide data for field-level EIQ calculations

**Import/Export Integration**

* Excel Integration: Models support serialization for Excel import/export
* Data Mapping: Repository classes assist in mapping external data to internal models
* Validation: Models provide validation for imported data integrity
* Format Support: Flexible data format support through model abstraction

**File System Integration**

* Resource Path: Repositories use resource\_path for consistent file access
* CSV Management: Direct CSV file reading with proper encoding and error handling
* Data Updates: Support for updating CSV data and refreshing repositories
* Build Integration: CSV files included in application packaging via PesticidesApp.spec

**Application Architecture**

The data folder serves as the foundation layer connecting:

* File System: CSV data files and resource management
* Business Logic: Model objects with validation and business rules
* Data Access: Repository pattern for consistent data access
* Application Layer: UI components, calculations, and user interactions

The data layer ensures the application has reliable, performant, and consistent data access while maintaining clean separation of concerns and supports the application's environmental impact assessment and pest management planning goals.

# common folder

## Purpose

The common folder serves as the foundational shared library for the application, providing reusable UI components, calculation engines, styling systems, and utility functions used throughout the application. It ensures consistency, maintainability, and reduces code redundancy across all application modules while centralizing core functionality like EIQ calculations, UI styling, and common widget behaviors. It provides a centralized “control room” to change styling, EIQ thresholds, and appearance in general.

## Components

**Core Modules**

1. constants.py - Application-wide constants:
   * EIQ threshold values for environmental impact classification
   * Regenerative agriculture framework thresholds
   * UI constants and configuration values
   * System-wide numerical constants and limits
2. styles.py - Centralized styling system:
   * McCain brand color definitions and styling functions
   * Font management with functions such as get\_medium\_font, get\_subtitle\_font
   * GENERIC\_TABLE\_STYLE for consistent table appearance
   * Widget styling functions and theme management
3. utils.py - Core utility functions:
   * resource\_path for file path resolution
   * get\_config for user preferences access
   * get\_eiq\_color and get\_eiq\_rating for EIQ visualization
   * get\_regen\_ag\_class for regenerative agriculture classification

**Calculation System (more on this in the dedicated chapter)**

1. calculations/layer\_1\_interface.py - Primary calculation interface:
   * eiq\_calculator main calculation engine
   * High-level calculation interface for UI components
   * Integration with user preferences and configuration
2. calculations/layer\_2\_uom\_std.py - Unit standardization layer:
   * Unit of measurement conversion and standardization
   * Rate calculation normalization for EIQ computation
   * Integration with UOMRepository and conversion systems
3. calculations/layer\_3\_eiq\_math.py - Core EIQ mathematics:
   * Environmental Impact Quotient calculation algorithms for applications of products with multiple AIs and scenarios with multiple applications
   * Mathematical formulas for environmental impact assessment

**Widget Components**

1. widgets/header\_frame\_buttons.py - Basic widgets:
   * HeaderWithHomeButton for consistent page headers
   * ContentFrame for standardized content layout
   * create\_button for consistent button styling
2. widgets/product\_selection.py - Product selection widget:
   * ProductSelectionWidget for product dropdown selection
   * Integration with ProductRepository for product data
   * Search and filtering capabilities within product selection
3. widgets/application\_parameters.py - Application parameter widgets:
   * ApplicationParamsWidget for pesticide application configuration
   * ApplicationRateWidget with rate and UOM selection
   * Integration with EIQ calculation system for parameter validation
4. widgets/uom\_selector.py - Unit selection widget:
   * SmartUOMSelector with context-aware UOM categories
   * UOMSelectionDialog for comprehensive unit selection
   * Base state management with BASE\_UOM\_TEXT
5. widgets/scorebar.py - EIQ visualization widget:
   * ScoreBar for visual EIQ impact representation
   * Color-coded impact levels with configurable thresholds
   * Integration with regenerative agriculture framework
6. widgets/tracer.py - Calculation tracer (commented out, but the code is still there):
   * calculation\_tracer for debugging calculation processes
   * Step-by-step calculation logging and visualization

## Workflow

**Calculation Engine Workflow (more on this in the dedicated chapter)**

1. Calculation Request: UI components request EIQ calculations via layer\_1\_interface
2. Parameter Processing: layer\_2\_uom\_std standardizes units and rates
3. Mathematical Computation: layer\_3\_eiq\_math performs core EIQ calculations
4. Result Processing: Results processed through utility functions for color coding and classification
5. UI Display: Results displayed using visualization widgets like ScoreBar

**Widget Integration Workflow**

1. Widget Creation: Application pages create common widgets with consistent styling
2. Data Binding: Widgets connect to data repositories and calculation systems
3. User Interaction: User input captured and validated through widget interfaces
4. Signal Emission: Widgets emit signals for data changes and user actions
5. Calculation Trigger: Data changes trigger EIQ recalculations through calculation layers
6. UI Update: Results update visual components with color coding and classifications

**Styling and Theme Workflow**

1. Style Definition: styles.py defines McCain brand colors and styling functions
2. Widget Styling: Common widgets apply consistent styling through style functions
3. Theme Application: Application-wide theme consistency through centralized style management
4. Dynamic Styling: Context-aware styling based on data values (EIQ colors, validation states)

## Data Flow

**Calculation Data Flow**

UI Input → Layer 1 Interface → Layer 2 UOM Standardization → Layer 3 EIQ Math → Results

    ↓               ↓                     ↓                    ↓                ↓

Parameters → Validation → Unit Conversion → Mathematical Computation → Visualization

**Widget Data Flow**

User Interaction → Widget Signals → Data Validation → Repos Access → Calculation Engine

                                      ↓                     ↓                  ↓

Configuration → Signal Emission → Data Updates → UI Refresh → Visual Feedback

**Styling Data Flow**

Style Defs. → Widget Creation → Theme Application → Dynamic Styling → Visual Consistency

        ↓                  ↓                 ↓                   ↓                 ↓

Constants → Functions → Widget Styling → Data-driven Colors → Brand Consistency

**Key Data Transformations**

* Unit Standardization: User input units → Standard calculation units → Result units
* EIQ Calculation: Product/AI data + Application parameters → Field EIQ values
* Color Mapping: EIQ numerical values → Color codes → Visual representation
* Classification: EIQ values → Regenerative agriculture categories → User guidance

## Key Features & Functionalities

**Advanced Calculation System**

* Layered Architecture: Three-layer calculation system for accurate and precise calculations
* Unit Conversion: Comprehensive UOM handling with user preference integration to increase ease for the users
* EIQ Algorithms: Scientifically based Environmental Impact Quotient calculations
* Tracing Support: Detailed calculation logging for debugging and transparency

**Comprehensive Widget Library**

* Consistent UI: Standardized widgets ensuring application-wide consistency
* Smart UOM Selection: Context-aware unit selection with category-based organization
* Product Integration: Seamless integration with product data and selection
* Visual Feedback: Score bars and color coding for intuitive data representation

**Robust Styling System**

* Brand Consistency: McCain corporate colors and styling throughout the application
* Responsive Design: Flexible styling that adapts to different content and screen sizes
* Data-Driven Styling: Dynamic styling based on calculated values and data states
* Font Management: Consistent typography with scalable font system

**Utility Functions**

* Resource Management: Cross-platform file path resolution for packaged applications
* Configuration Access: Centralized user preference and configuration management
* Classification Systems: EIQ rating and regenerative agriculture framework integration
* Color Management: Consistent color application based on environmental impact values

**Development Support**

* Calculation Tracing: Detailed logging for calculation debugging and verification
* Modular Design: Reusable components that can be easily integrated across the application
* Error Handling: Robust error handling and validation throughout common components (although this should be replaced or integrated with proper logging)
* Documentation: Comprehensive documentation for all common components

## Integration with the Rest of the App

**Application-Wide Integration**

* Universal Access: All application pages and components use common widgets and utilities
* Calculation Engine: Central calculation system used by all features
* Styling Consistency: All UI components inherit McCain brand styling from common styles
* Configuration: Centralized configuration access for user preferences and application settings

**Page-Level Integration examples**

* Home Page: Uses common widgets for navigation and layout consistency
* Products Page: Leverages product selection widgets and EIQ color coding
* EIQ Calculator: Direct integration with calculation layers and visualization widgets
* Season Planner: Uses application parameter widgets and EIQ calculation system

**Data Layer Integration**

* Repository Access: Common widgets integrate with data repositories for product and UOM data
* Model Integration: Widgets work with data models for type safety and consistency
* Validation: Common data validation logic shared across data input components
* Calculation Integration: Seamless integration between UI components and calculation engine

**Cross-Platform Integration**

* Resource Path: Ensures proper file access across different operating systems, and for both development state and .exe state
* Font Scaling: Responsive font system that works across different display sizes and densities
* Theme Support: Consistent theming that adapts to different system configurations
* Build Integration: All common components properly included in application packaging

**Development Ecosystem Integration**

* Code Reuse: Maximum code reuse through purposedly-designed common components
* Maintainability: Centralized logic for easier maintenance and updates
* Testing: Common components designed for testability and validation
* Documentation: Comprehensive documentation within the code supporting development and maintenance

The common folder serves as the backbone of the application, providing the essential shared infrastructure that ensures consistency, maintainability, and high-quality user experience across all application features while supporting McCain's commitment to sustainable agriculture through scientifically based environmental impact assessment.

# Field use EIQ calculation logic

## Basic concepts and abbreviations

* CP = Control Product (a pesticide)
* AI = Active Ingredient (a molecule)
* UOM = Unit Of Measure
* Application = an application of a CP, characterized by the application rate and its UOM
* Scenario = a list of applications of CP (the spray plan for a season)
* EIQ = the value assigned to an AI by Cornell University
* Field Use EIQ = the actual impact value, for one AI is calculated as such:  
  The field use EIQ of a CP with *n* AI is calculated by summation:  
  in most cases n = 1 or 2, and the maximum value is 4.

## Calculation process

The EIQ calculations replicates exactly what you would do as a human on paper:

1. Gather all your data (numbers and UOM)
2. Convert (standardize) all the UOM, make sure they make sense, and that the UOM algebra checks out. To obtain results in units (the ones used in the RegenAg framework) you need to have CP application rate, AI concentration, and AI EIQ in the following standard UOM:
   * Dry products:
   * Liquid products:
3. Once you have standardized all the UOM and checked that they make sense, you can execute the multiplication and get your result in the correct UOM.

**In the app**

The logic is contained in the “~/common/calculations/…” folder, and it is divided in 3 scripts, you can think of them as the parts of a handheld calculator: the first layer (keys and screen) gives you a simple interface to input data and read values from, while the internal layers (2 and 3) do all the math behind the scenes.

1. **layer\_1\_interface.py**  
   acts as the “interface” of the calculation logic for the rest of the app, providing 2 simple functions that are easily understandable when used in other parts of the code: calculate\_product\_field\_eiq and calculate\_scenario\_field\_eiq.
   * **calculate\_product\_field\_eiq**

It calculate the Field use EIQ for a product with up to 4 active ingredients. It takes as arguments: active\_ingredients (List of dictionaries with 'eiq', 'concentration', 'uom', 'name'), application\_rate (Product application rate), application\_rate\_uom (UOM of the application rate), applications (Number of applications), user\_preferences (User preferences for UOM conversions for seed treatments and CP applied as amount/unit of row length such as , which is common for CP applied in the furrow), and returns the Total Field EIQ in the correct UOM

* + **calculate\_scenario\_field\_eiq**

Calculate total Field EIQ for a scenario (multiple applications). It takes as arguments: applications (List of application dictionaries with product and rate info), user\_preferences (User preferences for UOM conversions) and returns the Total scenario Field EIQ in the correct UOM

From a technical point of view, it is created as a singleton instance of the object called “eiq\_calcualtor”, and used in other scripts like this:

from common.calculations.layer\_1\_interface import eiq\_calculator

… other code …

field\_eiq = eiq\_calculator.calculate\_product\_field\_eiq(

active\_ingredients = product\_data["active\_ingredients"],

application\_rate = product\_data["rate"],

application\_rate\_uom = product\_data["unit"],

applications = product\_data["applications"],

user\_preferences = user\_preferences

)

1. **layer\_2\_uom\_std.py**

Handles all UOM conversions and dimensional validation analysis, using intuitive functions such as “standardize\_product\_inputs”, “standardize\_single\_ai\_inputs”, “\_standardize\_application\_rate”, “\_standardize\_ai\_concentration”, etc.

1. **layer\_3\_eiq\_math.py**

Once all the UOM have been standardized and the values properly converted, this layer will perform the actual multiplication: it contains only 3 functions: “calculate\_field\_eiq\_single\_ai”, “calculate\_field\_eiq\_product”, and “calculate\_field\_eiq\_scenario”. The first (single\_ai) does the actual calculation for each AI, the second (product) just provides the data, calls the first multiple times and sums the results for all the AI of a product, and the third (scenario) calls the second for each product application in a scenario.

# Data used and how it was produced

The app in its current state uses 3 csv files as local databases:

* csv\_UOM.csv
* csv\_AI.csv
* csv\_products.csv

## UOM csv

This one I made myself, it is very simple and self-explanatory, here’s the full file:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| UOM | category | state | factor | standard |
| kg | weight | dry | 1 | kg |
| g | weight | dry | 0.001 | kg |
| lb | weight | dry | 0.453592 | kg |
| oz | weight | dry | 0.0283495 | kg |
| cwt | weight | dry | 50.8023 | kg |
| 100kg | weight | dry | 100 | kg |
| US ton | weight | dry | 907.185 | kg |
| metric ton | weight | dry | 1000 | kg |
| ha | area | no | 1 | ha |
| acre | area | no | 0.404686 | ha |
| fl oz | volume | liquid | 0.0295735 | l |
| pt | volume | liquid | 0.473176 | l |
| qt | volume | liquid | 0.946353 | l |
| gal | volume | liquid | 3.78541 | l |
| US gal | volume | liquid | 3.78541 | l |
| CA gal | volume | liquid | 4.54609 | l |
| l | volume | liquid | 1 | l |
| ml | volume | liquid | 0.001 | l |
| 100gal | volume | liquid | 378.541 | l |
| m | length | no | 1 | m |
| 100m | length | no | 100 | m |
| ft | length | no | 0.3048 | m |
| 1000ft | length | no | 304.8 | m |
| cm | length | no | 0.01 | m |
| inch | length | no | 0.0254 | m |

It can easily be expanded as needed.

## AI csv

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| NAME | type | EIQ | FRAC group code | HRAC group code | IRAC group code | notes |
| … | … | … | … | … | … | … |
| Cyazofamid | fungicide | 15.33333333 | 21 |  |  |  |
| Cycloxydim | herbicide | 19.33 |  | 1 |  |  |
| Cyfluthrin | insecticide | 31.56666667 |  |  | 3A |  |
| … | … | … | … | … | … | … |

This one was produced by printing a [report from GXCore](https://mccainag.lightning.force.com/lightning/r/Report/00OOF0000025lmj2AA/view) containing all the active ingredients, and then I added the type (all small letters consistently), picking from the types available in GXCore for the control products, avoiding the generic term “Biological” which means nothing, and using instead the functional type (e.g., a “biological fungicide” is a fungicide):

* adjuvant
* fumigant
* pgr
* fungicide
* herbicide
* insecticide
* rodenticide
* sanitizer
* fuel for ethylene generator
* molluschicide
* elicitant
* fertilizer
* amendment

I manually added the chemical classification and the EIQ values, two things I think MUST be added to GXCore. The EIQ values are the ones published by Cornell on the 1st of April 2024: these are the ones used to create the regen ag framework, they should be updated to the new ones when McCain adopts them.

The whole app right now uses names for matching products and AIs but this should be changed to use the proper IDs once it is connected to GXCore. In the products page and EIQ calculator page this is not an issue because the user clearly selects the name-app method, and the products are already filtered by region, but in the scenario planner this may result in slight errors where a product is matched to the wrong CP standard, this is not actually an issue for now as the AI concentrations are the same for all CP standards of a same CP, but I think this should be made more robust by using the proper CP IDs (I could not use them when I developed the app because I was not given access to the data initially, but it won’t be too hard adjusting the logic to use them).

Note that the AIs need some cleaning (this should happen at the GXCore level) as some are registered with slight synonyms, different capitalization, or just irrelevant stuff (e.g., “Mefenoxam”, “Mefenoxam (foliar)” and “Mefenoxam (soil)”: these are the same AI and the soil application is something that refers to a product use, not an AI!)

## Products csv

Column headers:

* country
* region
* type
* regulator number
* name
* registrant
* application method
* formulation
* min rate
* max rate
* rate UOM
* min days between applications
* REI (h)
* PHI (d)
* AI1 *(name, same for 2, 3, and 4)*
* [AI1] *(concentration, same …)*
* [AI1]UOM
* AI2
* [AI2]
* [AI2]UOM
* AI3
* [AI3]
* [AI3]UOM
* AI4
* [AI4]
* [AI4]UOM

This file was created merging two GXCore reports: [one as is for the control product standards](https://mccainag.lightning.force.com/lightning/r/Report/00OOF0000024E1d2AE/view?queryScope=userFolders), and [one that requires some transformation](https://mccainag.lightning.force.com/lightning/r/Report/00OOF000001wj4D2AQ/view?queryScope=userFolders) to bring all the active ingredients for a control product on a single row.

Here are the steps I followed as I initially did not have access to Salesforce (GXCore) raw tables, nor did I receive any introduction to data structures, so I had to rely on what I could find by myself: reports, but when connecting to GXCore the app should be adjusted to use the raw tables pulled, this will greatly simplify the data pulling procedure.

1. Refresh and download [this report](https://mccainag.lightning.force.com/lightning/r/Report/00OOF0000024E1d2AE/view?queryScope=userFolders) (I’ll call this “table 1” later)
2. Rename and reorder the columns to match the structure given above (keep the CP ID for now though!)
   1. Ensure the min rate and max rate columns are expressed in the same UOM (this is something that should be fixed at the GXCore level though…), converting where necessary, and keep only one rate UOM column
   2. Merge the “Minimum Days Between Applications” and “Standard Min Days Between Applications” columns to minimize the blanks
3. Ensure every row has a country (fill the blanks)
4. Either fix the UOMs to match the ones in the UOM CSV or add the synonyms to it (this should also be fixed at the GXCore level, UOMs should always be written consistently!)
5. Refresh and download [this report](https://mccainag.lightning.force.com/lightning/r/Report/00OOF000001wj4D2AQ/view?queryScope=userFolders) (I’ll call this “table 2” later)
6. Clean the file to keep only the raw table
   1. Delete rows 1 to 17
   2. Delete bottom rows (“Total”, Sum, Count, …)
   3. Delete column A and K
   4. Unmerge columns A and B
   5. Delete column B
7. Merge columns B and H, and assign the correct country to all the rows that are missing it
8. Transform the data so that for each CP ID there is only one row with these columns {country, name, region, AI 1, AI1 concentration, AI1 conc. UOM, AI 2, AI2 concentration, AI2 conc. UOM, … same for AI 3 and 4}
   1. I did this partly manually and partly with a little custom python script which I can no longer find unfortunately, I must have deleted it, this is a painful process and will greatly benefit from using the actual data structures and IDs matching from GXcore
9. Delete duplicate rows
10. Import both cleaned tables 1 and 2 in the same excel workbook
11. In table 1, create the {AI x, AIx concentration, AIx conc. UOM} (x=1,2,3,4) columns
12. Using the xlookup function on the CP ID columns, import the AI1, 2, 3, and 4 data from table 2 into table 1.