

IceCap & Portfolio



Index

Index	2
Introduction	5
Why you should Modernize your 5250 Programs	5
Key Features and Benefits	7
Technical Details	10
Product Features	15
Product Feature Summary	34
About us	39
Installation Guide	42
Requirements	42
Installing IceBreak	42
Installing Portfolio	42
Installing IceCap	43
Uninstalling	43
Getting started	45
Running Portfolio first time	45
Running IceCap first time	47
Setting up Portfolio	50
Setting up IceCap	51
Coding with IceCap	53
Interprocess Communication	57
IceCap	60
Presentation	60
Tags	64
Plugins	71
Charts and Documents	75
Integration	88
Virtual Terminal	138
Status bar	145
Translation	147
Start PC Organizer	150
Keystroke Buffering	153
Demo Environment	155
Portfolio	162
What is Portfolio?	162
Using Portfolio	163
System	176
Menu Setup	180
User Management	186
User Roles and Permissions	193

Customizing Portfolio	197
Portfolio Client Manager	202
The Notification Center	207
Configuration	209
IceCap Configuration	210
IceBreak Server Configuration	211
Presentation keywords	212
Emulation keywords	216
Plugins keywords	217
Operation keywords	218
IBM i Configuration	219
Appendix A - Implementation Process Template	221
The process	221
The products	222
Appendix B - Subfile options & Function keys	222
Subfiles Options	224
Functions Keys	224
Appendix C - Keyboard shortcuts	227
Windows and tabs	227
Forms	227
Operations	227
System Control	228
Customized Shortcuts	229
Appendix D - Virtual Terminal APIs	230
Appendix E - Web Components ('xtype')	237
Appendix F - Customized Menu-Loader	241
Building a Menu Structure	241
Appendix V - Operations and System Monitoring	257
Managing the Solution Stack	257
Monitoring and System Management	259
Appendix W - User Authentication and Data Integrity	262
Appendix X - Troubleshooting and Technical Support	265
Reporting Technical Issues	265
Tracing Issues in IceCap	265
Appendix Y - License Agreement	267
Appendix Z - RPG Source Repository	269
Configuration	271
Emulation for IceCap Flat	278
Emulation for IceCap Flex	281
Emulation for IceCap Flow/Flip	288
Interpretation	314
Navigation	363
Organization (PC Organizer)	365

Virtual Terminal API (VT)	367
---	---------------------

Introduction

Why you should Modernize your 5250 Programs

The IBM i platform has stood the test of time, delivering unmatched reliability, cost-efficiency, and security for over three decades. Its continued success is largely due to its bulletproof architecture and the essential role it plays in supporting critical IT infrastructures for thousands of organizations worldwide. However, one of the key elements behind IBM i's enduring presence is the 5250 green-screen applications, which were integral in meeting the enterprise demands of a previous era.

Despite their historical importance, these legacy applications are now outdated and present challenges in the modern digital landscape. Many organizations must now confront the limitations of 5250 green screens and recognize the necessity of modernization.

Why Modernizing your 5250 is Urgent

5250 green-screen applications were created in an era that predates the internet and modern web technologies. As a result, they lack web-enablement, an essential feature for today's IT systems. In today's internet-centric world, this shortcoming means that 5250 applications are ill-equipped to meet the demands of modern businesses.

In addition to the lack of web capabilities, these programs - written in languages like RPG and COBOL - suffer from outdated user interfaces, inefficient workflows, and functionality that has not kept pace with technological advancements. Modern businesses, focused on digital transformation, are finding it increasingly difficult to justify using tools that hinder agility and digital integration. Every year, the pressure to modernize these systems grows, as organizations realize they need to remain competitive and relevant.

Approaching 5250 Modernization with IceCap and Portfolio

The decision to modernize is not simple, especially for organizations that are concerned about the costs or disruptions associated with a complete system overhaul. Full-scale updates to legacy RPG and COBOL solutions are often viewed as too costly or complex for most companies to undertake.

However, IceCap and Portfolio offer a compelling solution to these challenges. With their comprehensive set of tools, these platforms simplify the modernization process, allowing companies of all sizes to transform their IBM i solutions without the need for extensive financial investment.

By utilizing IceCap and Portfolio, enterprises can transition their 5250 programs into a modern, browser-based environment. This streamlined transformation process not only updates the visual interface but also enhances workflows, improves usability, and enables organizations to integrate with modern IT infrastructures - all without altering the core applications. This approach makes modernization accessible and achievable for a broad range of businesses.

Benefits of Modernization with IceCap and Portfolio

One of the significant advantages of IceCap and Portfolio is that they can be implemented quickly - installed in hours, configured in days, and optimized within weeks. This reduces the traditional hesitation associated with modernization projects, such as concerns over disrupting established workflows or systems. Moreover, both products are designed to continuously adapt to the evolving needs of your business.

In short, IceCap and Portfolio present a practical and efficient solution for modernizing IBM i applications, addressing the need for web integration, user-friendly interfaces, and optimized workflows. These platforms are the ideal choice for businesses that want to future-proof their operations while preserving the core reliability of their existing IBM i solutions.

Modernization Guide

This guide offers comprehensive instructions on using IceCap and Portfolio to modernize your ERP system. By addressing the limitations of the traditional 5250 interface, IceCap and Portfolio transform your ERP into a modern, web-based solution, enhancing both its usability and functionality.

Key Features and Benefits

IceCap and Portfolio enable organizations to modernize IBM i applications by transforming traditional 5250 interfaces into a browser-based, web-enabled environment. The platform supports a comprehensive suite of features designed to enhance usability, improve workflow efficiency, and enable integration with modern IT systems.

By bridging the gap between legacy systems and modern user expectations, IceCap and Portfolio provide a powerful solution for businesses seeking to leverage their IBM i applications in a contemporary setting.

Modernized Presentation for Enhanced User Experience

IceCap and Portfolio convert the 5250 green-screen applications into web-enabled interfaces that are visually appealing and accessible from any device with a browser. This transformation overcomes many limitations associated with the traditional green screen, making applications more intuitive and user-friendly.

The platform includes customizable presentation modes and themes, allowing users to select from different interface styles that best fit their preferences and organizational branding. With these enhancements, IceCap and Portfolio deliver a modern, visually cohesive experience that aligns with user expectations in today's digital environment.

Streamlined Navigation and Multitasking Capabilities

Navigation is simplified through a structured menu system that organizes applications into an easy-to-navigate tree structure, supporting efficient access to applications and resources. IceCap and Portfolio support true multitasking, allowing users to open multiple applications simultaneously in separate tabs or windows, similar to modern web applications.

Users benefit from both mouse and keyboard shortcuts for rapid access to frequently used functions, reducing the time required to complete tasks and enhancing overall workflow efficiency. This streamlined navigation and multitasking capability enable users to handle complex tasks without excessive switching or redundant navigation steps.

Improved Workflow Efficiency and Optimized Data Entry

IceCap and Portfolio significantly enhance workflow efficiency by enabling custom field transformations and interactive input tools. Data entry fields across screens are consistently transformed, with validation rules improved to prevent data entry errors and reduce redundant tasks.

The platform introduces interactive tools such as combo boxes, checkboxes, radio buttons, and date pickers, which replace standard input fields, speeding up data entry and improving accuracy. Additionally, the solution allows users to consolidate multiple 5250 screens into a single workflow, minimizing repetitive tasks and enhancing task completion speed. These workflow optimizations enable users to achieve greater productivity while reducing potential errors.

Seamless Integration with Third-Party Systems for Enhanced Connectivity

The platform offers robust integration capabilities, creating a unified service layer that transforms 5250 programs into web services. This layer allows IBM i applications to integrate seamlessly with modern applications and third-party software, enhancing data accessibility and promoting collaboration across different departments or platforms.

IceCap and Portfolio support the embedding of 5250 programs within other applications and provide the ability to integrate charts, images, and documents from various data sources. By facilitating connectivity between IBM i applications and modern systems, IceCap and Portfolio help organizations improve cross-functional collaboration and ensure that critical data is accessible across platforms.

User-Friendly Onboarding and Enhanced Documentation

IceCap and Portfolio improve onboarding for new users by simplifying access to resources and adding intuitive navigation aids. The platform's Assist feature, a real-time search tool, reduces repetitive data entry and offers suggestions to guide users efficiently through fields and screens. IceCap also provides detailed tooltips, field explanations, and improved field labels to address gaps in legacy documentation and reduce confusion.

These features create a more intuitive environment for both new and experienced users, lowering the learning curve and facilitating efficient operation. Enhanced documentation and intuitive navigation tools contribute to a streamlined, productive user experience.

Technical Details

This chapter provides a technical overview of how IceCap and Portfolio integrate within the IBM i architecture and the existing application environment.

Designed to function as middleware, IceCap and Portfolio offer a robust, native solution for modernizing IBM i applications. By seamlessly embedding into the IBM i infrastructure without requiring changes to source or display files, the system maintains the integrity and stability of the original applications while introducing substantial enhancements in functionality and usability. This approach enables organizations to extend the capabilities of their IBM i applications within a modernized, web-enabled framework.

Technology

Nine core technological features make IceCap and Portfolio an ideal choice for modernizing IBM i applications:

1. **Middleware Implementation**

IceCap and Portfolio act as middleware, bridging IBM i applications and web interfaces. Operating between the original program and the browser, they work without needing access to source files or making changes to display files. This ensures that the base applications remain untouched, retaining their proven reliability.

2. **Native IBM i Compatibility**

Both IceCap and Portfolio run natively on IBM i, eliminating the need for additional servers or complex infrastructure. This native operation is key to delivering high performance and seamless integration, allowing organizations to leverage IBM i's robust capabilities directly within the platform's ecosystem.

3. **5250 Interface Integration**

IceCap and Portfolio support applications with traditional 5250 interfaces, including legacy programs written in RPG or COBOL, and are compatible with applications still operating in the S/36 environment. This broad compatibility allows organizations to modernize a wide range of IBM i applications.

4. **Authorization Compliance**

IceCap and Portfolio respect the IBM i authorization and security settings, mirroring the security setup of both the application and operating system. This approach ensures that user permissions and access controls remain consistent with established protocols, maintaining high standards of data security.

5. **Browser-Based Accessibility**

The solution provides access through a web browser, making applications available on any device with internet access. This web-based model promotes flexibility and accessibility, enabling remote work and multi-platform compatibility without requiring client-side installations.

6. **Dynamic Display Adaptation**

IceCap and Portfolio automatically adapt to changes in IBM i display files, ensuring that applications run smoothly even as display parameters evolve. This adaptability maintains uninterrupted functionality and supports ongoing application updates without manual reconfiguration.

7. **Configurable Components**

Both platforms offer extensive configurability, allowing organizations to adjust features and functions to meet specific needs. Users can customize elements of the interface, navigation, and workflow without altering the original applications, enhancing user experience while retaining the core structure.

8. **Tree Structure and Multitasking Support**

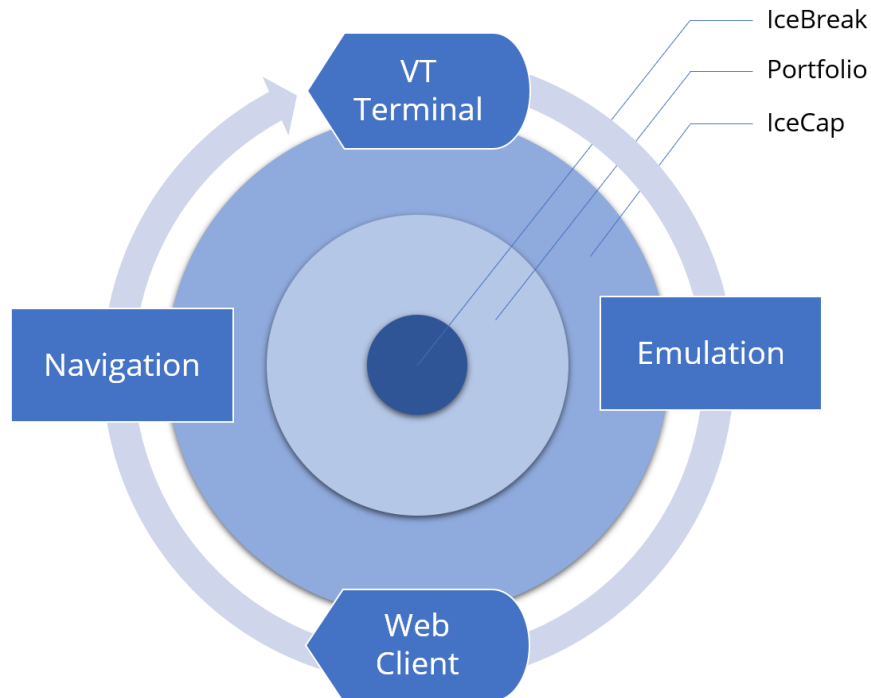
IceCap and Portfolio organize applications in a tree structure, providing a logical, hierarchical view that simplifies navigation. Additionally, they support multitasking with tabbed and windowed views, enabling users to run multiple applications simultaneously for efficient workflow management.

9. **Presentation Modes and Themes**

With four presentation modes (Flat, Flex, Flow, Flip) and eight color themes, IceCap and Portfolio provide a highly customizable user experience. These options allow organizations to tailor the appearance of their applications to align with branding requirements and user preferences, creating a cohesive and visually appealing interface.

Architecture

IceCap architecture is designed to modernize legacy ERP systems by transforming traditional programs into web applications or services, providing a robust and scalable framework for enhanced functionality and usability. The architecture of IceCap consists of several key components and processes that work together to achieve this transformation.



The key components interact in the following cycles: The Virtual Terminal (1) supplies the 5250 screens to the Emulation (2), which enriches and transforms the screen to be shown in the Web Client (3). The Web Client returns through the Navigation (4), which can alter the workflow before returning to the Virtual Terminal, ready for the next cycle. Further details on each component are provided below:

Virtual Terminal

In the IBM i operating system, a Virtual Terminal (VT) is a software-based emulation of a physical terminal. Virtual Terminals interact with the system, allowing users and applications to perform tasks as if they were using a physical terminal but without the need for physical hardware.

Many IBM i applications are designed to work with terminal-based interfaces. Virtual Terminals provide compatibility with these applications, ensuring that users can interact with software as intended. Today, Virtual Terminals serves as the middleware for all emulation software, such as IBM Personal Communications, Mocha TN5250, and other 5250 emulators.

Virtual Terminals, being subject to the same security measures as physical terminals, ensure the safety of your data. This includes user authentication, access controls, and auditing capabilities to ensure only authorized users can access and interact with the system.

This component integrates with the original programs, serves the 5250 terminal screens as a datastream, and ensures that all data is received and stored correctly.

Emulation

The Emulation component features a comprehensive toolset to manage the data stream from the Virtual Terminal. Before processing, the data stream is converted into a JSON structure, which can be accessed and manipulated via web service tools.

In its default configuration, the Emulation component prepares the datastream for a 1:1 presentation of the 5250 screen in the web client. The only modifications are extracting the screen title to display as the tab name and separating function keys (F-keys) to present them as interactive buttons.

Running natively on IBM i, the Emulation program has full access to all existing programs and files. Written in Free RPG, the program leverages the toolset to access the data stream, enabling RPG developers to modify existing 5250 programs to meet specific needs and provide a modern interface without altering the original programs.

For those without developer skills, IceCap provides numerous keywords and Tags that can be used to change and manipulate the visual appearance of the transformed application, giving you the power to customize your interface.

This component handles the emulation, interpretation, and language translation of the 5250 data stream.

Web Client

The web client component transforms the JSON-based data stream into a new web-enabled version of the 5250 screen. Using a comprehensive set of keywords and the Emulation component, the

presentation and behavior of the screen can vary from a 1:1 version of the 5250 to a full-blown web application.

Combined with Portfolio, the web client becomes a complete web-based system environment with session handling that operates across browsers, platforms, and hardware. Portfolio unites IBM i users with users from other platforms into one common solution, supporting user settings, roles, and permissions. The environment features an intuitive menu structure with a tab/window-driven application architecture. Additionally, an optional feature can generate new applications with a similar appearance to the 5250 Emulation and other applications in Portfolio.

This component presents 5250 programs as applications alongside other web and cloud-based applications in an intuitive multitasking system environment.

Navigation

The Navigation component intercepts the data stream going back to the virtual terminal. By running natively on IBM i, it opens up various possibilities to change the workflow without reprogramming existing programs.

Over the last 30 years, workflows have evolved significantly. Today, users often complain about the extra time it takes to enter several programs and retype key information just to complete a simple process. Modern users expect fewer programs with multiple functions rather than many programs with simple tasks requiring extensive menu operations and program shifts.

The Navigation component is designed to enhance workflows in the IceCap architecture. It can chain programs together and combine them with additional business logic, thereby simplifying complex processes. It can navigate through menus and programs in the background, enter and process data, and even respond to error messages. Essentially, it simulates the user's time-consuming semi-manual workflows, making the system more user-friendly and efficient.

This component enables new workflows based on existing programs initiated by the new UI or added business logic.

Product Features

IceCap and Portfolio integrate seamlessly to modernize traditional IBM i applications, transforming them into a web-based interface that improves user experience, workflow efficiency, and usability. This chapter details the product features and use cases offered by IceCap and Portfolio.

Application Organization and Navigation

IceCap's Portfolio module optimizes application organization and navigation, transforming traditional ERP structures into an intuitive, multi-tasking environment that enhances productivity. By grouping applications logically and using a tab-based interface, Portfolio simplifies access to frequently used systems and supports efficient task management.

Organized by System with Tree Structures

Portfolio categorizes applications into systems and organizes them within a hierarchical tree structure. This approach helps users locate and access the desired applications quickly, allowing them to navigate through complex ERP systems with ease. The organized structure supports users in finding applications based on function, department, or workflow, reducing search time and improving operational flow.

Tab-Based Interface for Multi-Tasking

Applications open in tabs within Portfolio, providing a clear overview of all active applications and ongoing tasks. This tab-based interface allows users to switch effortlessly between applications, supporting a multi-tasking environment that is essential for modern workflows. Users can move between tabs or open applications in new windows, enabling flexible navigation and making it easy to manage multiple tasks simultaneously.

Use Case: Streamlining Daily Operations for Customer Support

A company's customer support team uses a legacy ERP system with applications for managing customer profiles, tracking tickets, and accessing product information. Previously, the team had to navigate through a series of commands and screens, which made it difficult to switch between tasks and respond quickly to customer queries.

Solution with IceCap Portfolio's Application Organization and Navigation:

1. **System Division with Tree Structures:** Customer support applications are organized under a dedicated "Support" system in a hierarchical tree, grouping applications like "Customer Profiles," "Ticket Management," and "Product Lookup." This layout enables support staff to find relevant applications quickly without memorizing navigation paths.
2. **Tab-Based Multi-Tasking:** Each application opens in a separate tab, allowing representatives to have customer profiles, ticket management, and product information screens open simultaneously. Support staff can quickly switch between tabs when responding to inquiries, improving response time and customer satisfaction.
3. **Flexible Window Management:** Representatives handling complex cases can open specific tabs in new windows, arranging their workspace to view multiple applications side by side. This flexibility enhances productivity, as they can cross-reference information without repeatedly navigating back and forth.

This organized and flexible navigation system allows customer support representatives to efficiently handle multiple applications at once, improving response times and the overall customer experience.

IceCap's Portfolio module elevates traditional ERP navigation to a modern, efficient user experience that supports multi-tasking and enhances workflow.

Enhanced User Interaction Features

IceCap enhances user interaction by transforming traditional IBM i subfiles, function keys, and navigation into a streamlined, intuitive experience that resembles modern applications. This transformation improves productivity, reduces learning curves, and integrates seamlessly with both mouse and keyboard operations.

Subfiles Displayed as Interactive Lists

In IceCap, subfiles are presented as interactive lists where users can right-click to access available options directly. If a mouse is unavailable, users can press Control + Space to display the options

menu. The highlighted option serves as the default action when double-clicking a row, allowing for quick selection without additional input.

Streamlined Row-Specific Options

Instead of displaying a lengthy list of options at the top of the subfile, IceCap places the option field beside each row. This layout enables users to perform specific actions on multiple rows simultaneously, making bulk operations more efficient and enhancing control over list-based data.

Function Keys as Buttons for Improved Accessibility

IceCap displays function keys as buttons in a panel below the main screen, hiding F-numbers by default to create a cleaner, more user-friendly interface. Hovering over a button reveals the F-number, while pressing Control + F displays the full list of F-keys, allowing users to access these options with ease.

Integration of Mouse and Keyboard Shortcuts

The interface supports both mouse actions and keyboard shortcuts, with customizable shortcuts for frequently used programs. Tooltips display shortcut keys when hovering over buttons, icons, or tabs, providing a clear guide for users and enhancing workflow efficiency.

Support for Movable and Resizable 5250 Windows

IceCap maintains compatibility with most 5250 window variations, transforming them into genuine, interactive windows. These windows can be moved and resized, allowing users to customize their layout according to personal preference and enhancing the flexibility of their workspace.

Use Case: Enhanced Workflow for Order Processing

A retail company's order processing team frequently works with IBM i subfiles for managing and updating multiple customer orders. However, the traditional subfile setup and fixed function key layout make it challenging for new employees and slow down seasoned operators.

Solution with IceCap Enhanced User Interaction:

1. **Interactive Subfiles:** Orders are now displayed in a list format, allowing staff to right-click each order for specific actions, such as “Update Status” or “View Details.” This quick access to actions simplifies order processing and reduces the time spent navigating options.
2. **Row-Specific Options:** Option fields next to each order enable users to select and update multiple orders in one go, such as marking several orders as shipped or ready for invoicing, making bulk actions more efficient.
3. **Button-Based Function Keys:** Common functions like “Save” and “Cancel” appear as buttons below, with F-numbers revealed on hover for users familiar with keyboard shortcuts. This feature allows new employees to adapt quickly, while experienced staff can use familiar F-keys as needed.
4. **Customizable Shortcuts and Movable Windows:** Staff can set shortcuts for frequent tasks, and IceCap allows them to move and resize 5250 windows to create a personalized workspace. This flexibility makes multitasking easier and supports quick reference to other windows, such as customer records or product inventories.

This transformation of the order processing interface results in faster training for new employees, increased processing speed, and a more intuitive, adaptable workspace for experienced users.

IceCap’s revitalized user interface transforms traditional IBM i interactions into a modern experience, enhancing efficiency, usability, and adaptability.

Enhanced Data Presentation

IceCap’s Enhanced Data Presentation feature allows users to view comprehensive, structured information and perform data entry with advanced, intuitive tools. By consolidating relevant data and incorporating interactive elements, IceCap helps users make more informed decisions, improves data accuracy, and enhances workflow efficiency.

Comprehensive Information Display

IceCap enables users to display diverse information on a single screen, reducing the need to navigate between multiple screens. For example, an HR manager can view an employee’s salary

structure in a pie chart that visually breaks down salary, commission, and bonuses. This consolidated view helps users gain insights quickly and makes complex data easy to understand.

Additional Tabs for Streamlined Access

To improve navigation and access to related data, IceCap supports the addition of extra tabs within forms. This feature allows users to view relevant information without leaving the current screen. For instance, while processing employee data, a manager can switch to a tab to search for external information on a new employee or view a scanned version of their signed job application. These tabs streamline access to essential resources, speeding up decision-making and reducing context-switching.

Standard Data Entry Tools

IceCap enhances daily data entry by providing familiar tools such as combo boxes, checkboxes, radio buttons, date pickers, and tooltips. These tools simplify the entry process and improve accuracy. IceCap also allows users to display data from multiple 5250 screens in a single view, enabling users to enter and cross-reference data more efficiently.

Advanced Search Functionality

While traditional F4-Prompt remains available for searching valid values, IceCap's combo box feature provides a more integrated search experience without requiring users to leave the screen. The combo box can even offer generic search capabilities, helping users find options quickly and accurately within the current view, which significantly enhances data entry efficiency.

Use Case: Streamlined Data Management in Employee Onboarding

An HR department is responsible for onboarding new employees and frequently needs to access and update various types of data, such as compensation, job roles, and employment history. Traditionally, these details were spread across multiple screens, and HR personnel had to switch between different applications to retrieve related documents.

Solution with IceCap Enhanced Data Presentation:

1. **Unified Compensation View:** With IceCap, HR staff can now see an employee's salary breakdown in a pie chart on the same screen as other employee information. This allows

for an immediate understanding of total compensation without needing additional calculations.

2. **Additional Tabs for Documentation:** While entering new employee details, HR can switch to a tab to search for related information, such as background checks or prior employment documents, directly from the onboarding form. This simplifies data verification and document retrieval, saving time.
3. **Standard Data Entry Tools:** The HR team uses combo boxes to select job roles and date pickers for employment start dates, reducing the chance of input errors and creating a smoother onboarding process.
4. **Enhanced Search:** The combo box feature allows HR staff to search for department codes and job titles within the same screen, offering faster and more efficient selection without the need to leave the data entry view.

This setup provides a seamless and efficient workflow for the HR department, allowing them to handle onboarding tasks in a single interface, improving accuracy, and reducing time spent navigating between screens.

IceCap's Enhanced Data Presentation enables organizations to streamline complex workflows, making data entry and retrieval more intuitive and efficient.

Special features

Reducing Repetitive Data Entry

Repetitive data entry can lead to inefficiency, errors, and user fatigue, especially when users repeatedly input similar information, such as names, addresses, and job titles. IceCap addresses this issue with its Assist feature, which minimizes keystrokes and improves accuracy by providing intelligent, real-time suggestions directly within data entry fields.

The Assist Feature: Real-Time Suggestions

The Assist feature offers smart, context-sensitive suggestions as users type into specific fields. When Assist is enabled on a field, a small blue triangle appears in the corner, signaling that suggestions are available. As users type, Assist performs a real-time database lookup, displaying potential values

along with their frequency of use, making it easier to select commonly entered data. This lookup is performed within the field itself, independent of the browser, form, or user settings.

If the desired value is not among the suggestions, users can continue typing, and Assist will save the new entry for future suggestions. For example, when entering an address, users can select a suggested street name and then manually enter the street number, reducing the overall typing required.

Duplicate Prevention

Assist also helps prevent duplicate entries by alerting users to similar existing records. When entering data such as company names, Assist checks for similar entries, reducing the likelihood of creating duplicate records and ensuring data consistency. This is particularly useful for fields that commonly contain repeatable or standardized information, like customer or product names.

Advanced Search Functionality

Beyond basic text matching, Assist can perform advanced searches by combining multiple fields in the search criteria. For example, a zip code search can be refined to show options only within a specified region, state, or country, helping users select the most relevant data for their context and reducing errors in address or location fields.

Use Case: Streamlining Customer Data Entry in Sales

A sales team frequently enters customer information into an IBM i-based CRM system. Often, sales representatives need to input common data, such as company names, addresses, and contact roles, which can be repetitive and error-prone. Additionally, duplicate customer records occasionally create confusion, slowing down customer interactions.

Solution with IceCap Assist:

1. **Real-Time Suggestions:** When entering the company name or contact title, Assist suggests frequently used entries (e.g., "Sales Manager" or "Product Support"). This reduces typing time and standardizes entries across the team.
2. **Duplicate Prevention:** As the representative types a company name, Assist flags any similar existing records, helping the user avoid duplicate entries and ensuring they are adding data for a truly new customer.

3. **Refined Searches for Addresses:** For address fields, Assist offers a dropdown with possible zip codes filtered by city or region, allowing the representative to select the correct information quickly and accurately.

This streamlined data entry process reduces redundancy, enhances accuracy, and prevents errors, helping the sales team maintain a clean, efficient CRM system and improving their response times with customers.

The Assist feature within IceCap simplifies data entry, increases accuracy, and helps maintain data quality in legacy IBM i systems.

System Transformation

IceCap's System Transformation feature enables organizations to modernize the visual interface of IBM i applications, providing a refreshed, user-friendly look while preserving the underlying functionality and reliability of the original system. By transforming only the presentation layer, IceCap ensures that IBM i screens align with modern user experience standards, all without altering the core application or requiring access to its source code.

Modernized Screens with SAA Compliance

IceCap transforms IBM i screens to align with IBM's Systems Application Architecture (SAA) standards, creating a familiar and updated user experience for today's workforce. This visual transformation makes IBM i applications more accessible and easier to navigate, while retaining the layout and structure that experienced users recognize. The enhanced interface provides a contemporary look and feel, bridging the gap between traditional functionality and modern expectations.

Presentation-Only Changes: "Integration on the Glass"

IceCap operates exclusively at the presentation layer, a method referred to as "Integration on the Glass." This approach modifies only the visual appearance of IBM i screens without altering the core

operating system, application logic, or ERP solution. As a result, organizations can enjoy a modern interface without the risk of disrupting established workflows, data integrity, or business processes.

Source Code Independence

One of IceCap's key advantages is its ability to operate independently of the source code. It functions as a virtual layer between the IBM i program and the browser interface, providing security and ease of deployment without requiring direct access to the program's codebase. This source code independence allows organizations to modernize their applications even if the source code is unavailable or restricted, making IceCap an efficient and flexible modernization tool.

Use Case: Enhancing Usability in a Manufacturing ERP System

A manufacturing company uses an IBM i-based ERP system for inventory and production management. While the system is robust and reliable, its outdated green-screen interface makes it difficult for new employees to learn and navigate efficiently. The company wants to improve usability without disrupting the core application or altering the source code.

Solution with IceCap System Transformation:

1. **Modernized Screen Presentation:** IceCap updates the ERP screens to a modern, SAA-compliant interface, making the system more intuitive and visually appealing. The screens retain the structure familiar to experienced users, while the refreshed look enhances ease of use for newer employees.
2. **Presentation-Only Changes:** Since IceCap only modifies the presentation layer, the system's core functionality remains unchanged. This "Integration on the Glass" approach allows employees to benefit from a modern interface while maintaining existing workflows and processes.
3. **Source Code Independence:** IceCap operates independently of the ERP's source code, so the transformation is implemented seamlessly without accessing or modifying the application's underlying codebase. This source code independence is ideal for companies relying on legacy systems with limited access to original code.

By modernizing the presentation layer, the company achieves a user-friendly interface that reduces training time for new employees and improves the overall user experience, all without disrupting its reliable ERP system.

IceCap's System Transformation provides a streamlined, modern user experience for IBM i applications while preserving the integrity and security of the underlying system.

Field Transformation

IceCap's Field Transformation feature with Assist enhances data entry by providing intelligent, real-time suggestions. Through tagging and rule-based configurations, Assist enables users to quickly find and select relevant data, such as zip codes and corresponding regions or cities. This feature reduces input errors, improves efficiency, and provides a more intuitive interface for data entry.

Setting Up Assist for Linked Field Suggestions

By tagging fields like "Zip Code" and "Region/City," Assist can be configured to display relevant suggestions as the user types. For example, when entering a zip code, Assist narrows down a list of possible values based on the initial characters, allowing the user to quickly find the correct entry. If the region or city field is updated, the zip code suggestions dynamically adjust to match the new selection, maintaining linked, contextually relevant options. These fields remain associated as long as they are positioned on adjacent lines in the interface.

Defining Rules for Contextual Assistance

Rules can be created to activate the Assist feature whenever specific labels—such as "Zip Code" or "Region/City"—appear on the screen. These rules are applied system-wide, ensuring that the Assist feature is available on any screen where these fields are displayed. Additionally, these rules can be triggered multiple times on the same screen, allowing users to benefit from Assist across different instances of these fields.

Enhanced Prompt Functionality for Legacy Fields

IceCap brings modern F4-prompt functionality to fields that traditionally lacked dynamic data suggestions, simplifying the user experience. By providing real-time assistance for fields like "Zip Code" and "Region/City," IceCap offers a functionality upgrade for IBM i applications, addressing a long-standing limitation.

User-Friendly Configuration and Demo Environment

Assist rules are easy to configure using RPG for rule logic and SQL for database access. This straightforward setup is supported by a complete demo environment, allowing users to experiment with and understand the Assist feature before implementing it in a production environment.

Use Case: Simplifying Address Entry for Customer Service

A logistics company's customer service team frequently enters address details into an IBM i-based application for shipment scheduling. To expedite address entry and reduce errors, the team needs an efficient way to select correct zip codes and region/city names, particularly in areas with complex postal codes.

Solution with IceCap Assist:

1. **Tagging Zip Code and Region/City Fields:** The zip code and region/city fields are tagged to enable Assist suggestions. As a user begins typing a zip code, Assist displays matching codes, helping them quickly locate the correct entry.
2. **Dynamic Updates:** If the user changes the region or city field, Assist dynamically adjusts the zip code suggestions to ensure only valid, contextually relevant options are available, reducing the risk of mismatches.
3. **System-Wide Rule Application:** A global rule is established to activate Assist for zip code and region/city labels on any screen, ensuring consistent support for address entry across the customer service application.

This setup enables customer service representatives to enter address information accurately and efficiently, streamlining data entry and reducing input errors, which is particularly beneficial in complex postal regions.

IceCap's Assist functionality within Field Transformation provides a modern, user-friendly approach to data entry, helping organizations improve accuracy and efficiency in legacy IBM i systems.

Presentation Modes

IceCap provides four unique presentation modes to suit various modernization needs for IBM i applications. Each mode offers a different level of transformation, enabling organizations to choose the interface style that best aligns with user requirements and workflow preferences.

Overview of Presentation Modes

1. **IceCap Flat:** Offers a direct, one-to-one web emulation of the 5250 screen, retaining the original layout and interface. This mode is ideal for users who are accustomed to the traditional 5250 experience and need minimal interface changes.
2. **IceCap Flex:** Provides a visually updated look with proportional fonts and minor aesthetic enhancements, modernizing the interface without altering functionality. This mode retains the familiarity of the 5250 interface while offering a more user-friendly, visually refreshed display.
3. **IceCap Flow:** Reimagines the 5250 interface as a web-like application, displaying subfiles as tables and reorganizing function keys into menus. Flow mode is beneficial for users who prefer a more intuitive, web-centric interface while maintaining essential 5250 functionality.
4. **IceCap Flip:** Transforms the entire system into a Windows-like interface, using tabs and interactive buttons instead of traditional function keys. This mode provides a fully modernized look and feel, closely resembling desktop applications, making it ideal for users who are less familiar with 5250 screens.

Use Case: Multilevel Interface Modernization in a Finance Department

A finance department uses an IBM i application to manage invoices and financial records. The team comprises both experienced IBM i users and new employees unfamiliar with the 5250 interface. By implementing different IceCap presentation modes, the department can meet the needs of all user groups.

Solution with IceCap Presentation Modes:

1. **Experienced Users:** Senior staff accustomed to 5250 screens use **IceCap Flat** for invoice processing tasks, allowing them to maintain the familiar screen structure while accessing it through a web interface.

2. **Mid-Level Staff:** For tasks like budget approvals and reports, staff members use **IceCap Flex**, which offers a clean, updated layout without changing the screen logic, making it easier to read while preserving functionality.
3. **New Employees:** Junior employees working on data entry and record updates use **IceCap Flow**, which presents the system as a web application with tables and reorganized menus, simplifying navigation for those who prefer a web-like experience.
4. **Department Heads:** For managers reviewing financial summaries and complex reports, **IceCap Flip** provides a Windows-like interface, making the data more accessible and familiar to users who do not regularly interact with IBM i applications.

This multi-mode approach enhances the efficiency of each user group by aligning the interface with their familiarity and preferences, improving productivity across the department.

IceCap's flexible presentation modes allow organizations to deliver a modernized IBM i experience that caters to various user groups, improving workflow and interface usability without compromising on core system functionality.

Menu System Integration

Integrating ERP Menu Structures

Portfolio allows organizations to integrate their existing ERP menu structures into a cohesive, tree-based navigation system. With this functionality, users can access familiar ERP application menus in a modern interface. IceCap includes a customizable RPG program that adapts the traditional menu system, displaying it as a structured, hierarchical tree within Portfolio, simplifying navigation for users accustomed to the legacy ERP menu format.

In addition to integrating ERP menus, IceCap provides a program to display the full IBM i menu system within Portfolio. This capability offers users the flexibility to not only access standard IBM i menus but also add custom programs and commands manually, tailoring the menu to their specific needs.

Use Case: Unified Interface for Order Processing

A retail company's customer service team uses a legacy ERP system on IBM i to process orders. However, the ERP menu system, with its sequential commands and limited navigation, slowed down order processing, especially for new employees unfamiliar with the command structure.

Solution with Portfolio Menu Integration:

1. **Tree-Structured ERP Menus:** The legacy ERP's order processing menu is integrated into Portfolio as a tree structure. This transformation organizes menu options by function, allowing customer service representatives to intuitively locate and access order processing tasks.
2. **Custom Program Access:** Frequently used custom order management programs are added to the menu manually, allowing representatives to quickly access specialized tools without navigating multiple command screens.
3. **Full IBM i Menu Integration:** The entire IBM i menu system is also available, enabling power users to access system administration tools without switching environments.

This integration results in a streamlined, efficient menu system that accelerates order processing, reduces navigation time, and provides a consistent interface for both new and experienced employees.

By integrating menu structures with Portfolio, organizations enhance the usability of ERP systems on IBM i, creating a cohesive and efficient navigation experience that accommodates legacy workflows while providing a modern interface.

Workflow Adaptation

Resolving Outdated Labels and Improving Data Validation

As business workflows and data requirements have evolved, users often encountered fields that no longer aligned with the information they needed to enter. This misalignment led to the unintentional misuse of irrelevant or outdated fields, as users attempted to repurpose fields without clear guidelines. Consequently, many field labels became misleading, and validation constraints were

inadequate to prevent errors. The lack of up-to-date documentation only added to the confusion, forcing users to rely on informal knowledge sharing and leading to inconsistencies in data entry.

Updating Field Labels for Accurate Representation

With IceCap's Translation feature, administrators can update field labels to accurately reflect the current use of each field, providing clarity for users and ensuring data consistency. Additionally, the Translation feature supports multiple languages, allowing the organization to adapt field labels for multilingual environments. For instance, a field originally labeled "Education Level" can be updated to "Salary Level" if it is now being used to capture compensation data, aligning the label with the actual data entry requirements.

Improving Data Validation for Accurate Input

IceCap's Tag feature offers powerful customization options for establishing validation rules for fields. These rules ensure users enter valid data in the correct format, reducing errors and enhancing the accuracy of entries. For example, if a field initially labeled "Education Level" accepted a numerical range (e.g., 16 to 20), it can be updated to display a combo box with clear options like "High School," "Bachelor's," and "Master's," providing users with a more intuitive selection process that aligns with current data requirements.

Use Case: Employee Management System

In an employee management application, an outdated field labeled "Education Level" was originally designed to accept a numeric code (16–20) representing education qualifications. However, over time, the organization began using the field to record employee job levels, creating confusion among HR staff and resulting in inconsistent entries.

By using IceCap's Translation and Tag features, the administrator can:

1. **Relabel the Field:** Update "Education Level" to "Job Level," ensuring the label clearly reflects the field's current purpose.
2. **Implement Validation:** Define a combo box with valid job levels (e.g., "Junior," "Mid-Level," "Senior"), allowing users to select from standard options rather than entering freeform codes.

With these changes, HR personnel can easily identify the field's purpose and select the appropriate job level, eliminating ambiguity and ensuring consistent entries across the employee management system.

By enhancing field labels and validation, IceCap aligns legacy systems with current data standards and workflows, reducing errors and improving user experience.

Translation

Addressing Multilingual Requirements

Globalization has made it increasingly important for ERP solutions to support multiple languages to accommodate diverse user bases and regional requirements. However, many legacy systems lack built-in multilingual support. IceCap addresses this need by providing robust translation capabilities, allowing organizations to easily modify and translate field labels and screen text. Additionally, IceCap enables precise control over translation scope, allowing translations to be applied to specific screens or across the entire system as needed.

With IceCap's Translation feature, organizations can create a user-friendly, multilingual interface without altering the core application code, making it adaptable to a variety of languages and regional settings.

Use Case: Multilingual Inventory Management System

A multinational manufacturing company operates an inventory management system on IBM i, where employees across various regions access the application in their native languages. The system was originally developed with English labels, which posed challenges for non-English-speaking employees.

Solution with IceCap Translation:

1. **Custom Translation by Screen:** For high-usage screens accessed by employees in specific countries, IceCap is used to translate field labels and descriptions into relevant local

languages, ensuring employees understand the application's functions and reducing the chance of data entry errors.

2. **System-Wide Translation:** For fields that are universally used, such as "Quantity" or "Item Code," IceCap applies translations across the entire system, so these labels appear consistently in the selected language across all screens.
3. **On-Demand Language Switching:** Users can switch between languages on demand, enabling multilingual employees to access the system in their preferred language. This flexibility is particularly beneficial for training new employees in their native language.

By implementing IceCap's Translation feature, the company enhances usability and accuracy across its global workforce, resulting in a smoother, more accessible interface for all employees.

IceCap's Translation feature empowers organizations to create a truly multilingual ERP environment, improving accessibility and efficiency for global teams.

Integration

IceCap's integration capabilities allow organizations to embed, extend, and streamline IBM i workflows by connecting with modern web applications, third-party software, and custom solutions. These integration options enable users to access external data sources and functionalities directly from IBM i screens, enhancing efficiency and facilitating seamless workflows.

Embedding Web Applications for Direct Access

IceCap enables organizations to embed other web applications within the IBM i environment, including IceCap itself. By embedding applications as windows, users can pass key values using a query string, providing immediate access to specific data or functionalities within the integrated web application. This setup is beneficial for applications that require real-time data from external sources or need to launch web-based tools directly from an IBM i session.

Advanced Scripting and Data Interaction

Through menu-driven scripting, IceCap allows users to automate interactions with 5250 screens, offering streamlined access to specific screens and updating IBM i data from third-party

applications. This feature is ideal for systems that frequently need to exchange data with external applications, providing a seamless interface for data updates, retrieval, and display from outside sources.

Enhanced Workflows Through Combined Screen Access

By combining select screens from 5250 programs with third-party applications, IceCap supports the creation of integrated workflows. Users can transition between IBM i applications and external software without switching contexts, increasing workflow efficiency and improving data continuity. For example, employees could view customer details from a CRM system alongside order processing data on a single screen, enabling quick access to relevant information across platforms.

Integration with Model Studio for Dynamic Interfaces

Using Sitemule Model Studio, organizations can develop fully customized web applications to replace fixed subfiles in 5250 programs. By embedding a flexible, dynamic web application within the traditional 5250 interface, users retain the original data entry and business logic while benefiting from a modern web interface. This integration achieves a hybrid approach, leveraging the stability of the existing IBM i logic while offering a web interface with enhanced user interactions.

Use Case: Integrated Customer Service Portal

A company's customer service team requires quick access to both IBM i order processing and customer profile data from a CRM system. By integrating these applications with IceCap:

1. **Embedding CRM Application:** The CRM is embedded as a window within the IBM i session. Customer service representatives access customer profiles by passing the customer ID in a query string, instantly retrieving relevant CRM data.
2. **Scripting for Data Updates:** Scripts are set up to automate updates to the order processing screen based on data changes in the CRM. This ensures that any changes to customer information are reflected in real-time on the order screen.
3. **Model Studio Interface:** Sitemule Model Studio is used to create a custom interface for the order processing subfile, which allows representatives to view order history in a more intuitive, web-based layout while preserving the original IBM i business logic.

This integrated solution allows customer service representatives to efficiently manage orders and customer profiles from a single interface, improving response time and ensuring data consistency.

By combining IceCap's application integration capabilities with IBM i's existing workflows, organizations can modernize data access, automate data interactions, and improve the user experience.

Product Feature Summary

Application Organization and Navigation Features

System Division	Organizes applications into systems with tree structures for easy navigation.	Quick access to customer support applications.
Tab-Based Interface	Opens applications in tabs, supporting seamless multi-tasking and task management.	Switching between customer profile and ticket screens.
Flexible Window Management	Allows tabs to be opened in new windows for customized workspace layouts.	Viewing customer profiles and tickets side by side.

Enhanced User Interaction Features

Interactive Subfiles	Presents subfiles as lists with right-click options for easy access.	Quick actions for individual customer orders.
Row-Specific Options	Adds options next to each row for efficient multi-row operations.	Bulk updates on multiple customer orders.
Button-Based Function Keys	Displays function keys as buttons, with F-numbers hidden by default.	Easy access to Save, Cancel, and other functions.
Mouse and Shortcut Integration	Supports both mouse actions and customizable keyboard shortcuts for quick access.	Tooltips and shortcuts for frequent order processing.

Movable 5250 Windows	Enables movable, resizable windows for customized layouts.	Personalized window arrangement for multitasking.
-----------------------------	--	---

Enhanced Data Presentation Features

Comprehensive Information Display	Displays consolidated data on one screen, like pie charts for compensation breakdowns.	Unified view of salary, commission, and bonus.
Additional Tabs	Adds tabs within forms to access related documents or external information.	Quick access to employee documents during onboarding.
Standard Data Entry Tools	Provides familiar tools like combo boxes, checkboxes, and date pickers for efficient data entry.	Simplified entry of job roles and employment dates.
Advanced Search Functionality	Offers in-screen search capabilities via combo boxes, enhancing data selection without screen changes.	Rapid selection of department codes and job titles.

Assist Feature Benefits

Real-Time Suggestions	Provides frequent data suggestions in real-time to reduce repetitive typing.	Quick entry of commonly used titles or company names.
Duplicate Prevention	Alerts users to similar existing records, reducing duplicate entries.	Preventing duplicate customer entries in CRM.
Advanced Search	Combines multiple fields in search criteria, refining options based on region, state, or country.	Region-specific zip code suggestions for addresses.

System Transformation Features

Modernized Screens	Transforms IBM i screens to comply with SAA standards, offering a familiar yet updated interface.	SAA-compliant interface for manufacturing ERP.
Presentation-Only Changes	“Integration on the Glass” approach updates the interface without changing core system functionality.	Preserves original workflows in a modernized display.
Source Code Independence	Operates independently of source code, acting as a secure layer between the program and browser.	Seamless transformation for systems with restricted code.

Field Transformation Features

Assist Setup	Tags fields for dynamic suggestions based on input, linking zip codes and region/city fields.	Streamlined zip code and city entry in customer service.
Rule-Based Activation	Activates Assist based on specific labels, applied globally across screens where labels appear.	Consistent support for address entry fields.
Enhanced Prompt Functionality	Adds real-time data suggestions to legacy fields, offering a modern data entry experience.	Prompt feature for zip code fields in address entry.

Presentation Modes

IceCap Flat	Direct 5250 web emulation, ideal for users familiar with traditional screens.	Senior staff processing invoices.
--------------------	---	-----------------------------------

IceCap Flex	Enhanced appearance with minor updates, retaining original 5250 layout and functionality.	Mid-level staff working with budget reports.
IceCap Flow	Web-like transformation with tables and menus, suitable for users preferring web applications.	Junior employees in data entry.
IceCap Flip	Windows-like interface with tabs and buttons, for users needing a fully modernized experience.	Managers reviewing financial summaries.

Menu System Integration Features

ERP Menu Integration	Adapts and displays existing ERP menus in a tree structure for intuitive navigation.	Integrating order processing menus for customer service.
Custom Program Addition	Allows manual addition of frequently used programs and commands to the menu.	Adding custom order management programs for quick access.
Full IBM i Menu Access	Enables access to the complete IBM i menu system within Portfolio for comprehensive functionality.	Accessing system tools alongside ERP menus.

Workflow Adaptation Features

Translation	Updates field labels to align with current data entry needs, supporting multilingual labels if required.	"Education Level" changed to "Job Level."
--------------------	--	---

Tag Validation	Allows definition of validation rules to display valid options, such as a combo box, ensuring accurate data entry.	Numerical range converted to job levels
-----------------------	--	---

Translation Features

Customizable Translation	Allows translation of labels at the screen level or system-wide.	Translating specific fields for high-use screens.
Language Flexibility	Supports multiple languages, meeting diverse regional needs.	Switching between English and local languages on demand.
Consistent Labeling	Applies universal translations for commonly used fields, ensuring consistency.	"Quantity" and "Item Code" labels across all screens.

Integration Options

Embedding Web Applications	Embeds other applications with query strings for dynamic access.	Embedding a CRM application in IBM i.
Scripting & Data Interaction	Automates data updates and retrieval from third-party applications.	Syncing CRM customer updates with IBM i.
Combined Workflows	Integrates 5250 screens and external apps for continuous workflows.	Viewing CRM data alongside order processing.
Model Studio Integration	Replaces 5250 subfiles with flexible web applications, retaining business logic.	Custom web interface for order history.

About us

System & Method (System & Metode A/S) is a software development company founded in 1989, headquartered in Denmark, with an additional office in Brazil. We have a strong focus on developing solutions that modernize and optimize legacy systems on IBM Power Systems, particularly on the IBM i (AS/400) platform.

Key Offerings and Expertise

System & Method specializes in creating software that allows businesses to maximize the efficiency of their legacy IT infrastructure. The company's products aim to address the challenges of using traditional, terminal-based IBM i applications by transforming them into modern web-based solutions. Some of their flagship solutions include:

1. **IceBreak:** A native web and application server for RPG and COBOL, facilitating web-based application development on IBM i.
 2. **IceCap:** A solution specifically designed to modernize traditional "green screen" 5250 interfaces by transforming them into web-enabled applications without changing the core application code.
 3. **Portfolio:** Works with IceCap to create a cohesive, multitasking environment that allows users to run IBM i applications alongside other web applications, unifying the interface and supporting enhanced workflows.
 4. **Sitecule Architect:** Part of System & Method's suite for web-based application development, allowing seamless integration of traditional IBM i applications with modern, flexible interfaces.
-

Philosophy and Approach

System & Method focuses on aligning technology with business goals, ensuring that its products not only modernize legacy systems but also enable businesses to future-proof their operations. Our approach emphasizes:

- **Integration without Disruption:** Their products, like IceCap and Portfolio, allow legacy systems to be transformed without altering underlying code or disrupting established business processes. This minimizes costs and downtime associated with modernization.
 - **Flexibility and Customization:** With various configuration options (such as keywords, tags, and plugins), the company's solutions can be tailored to meet specific business needs, enhancing usability and productivity.
 - **Long-term Compatibility:** By using a middleware approach, System & Method's products are designed to remain compatible with evolving web standards and IBM i updates, ensuring sustainable modernization.
-

Industry Impact and Global Reach

System & Method is well-established in the Nordic region and has expanded globally, serving diverse industries and sectors. The company's solutions are particularly valuable for organizations reliant on IBM i systems in industries such as finance, manufacturing, retail, and logistics, where reliability and operational continuity are crucial.

Additional Offerings and Initiatives

In addition to their main products, System & Method provides other software tools that complement their modernization suite, including:

- **BlueSeries and BlueNote:** Tools for enhancing business communication (like Fax, Mail, SMS, FTP) management, streamlining data exchange, and improving system notifications.
- **Open-source Engagement:** The company also participates in various open-source initiatives, contributing to broader software development communities.

Through our comprehensive approach to modernization, we have helped numerous organizations update their IBM i applications to meet modern business needs, offering a path for future developments without losing the reliability and security IBM i systems are known for.

Installation Guide

Requirements

The following requirements have to be met:

- IBM i OS 7.2 or newer
- Access to a user profile with SPCAUT(*ALLOBJ *SECADM *JOBCTL)
- SSH must be started. If not, run the command: STRTCPSVR *SSHD (port 22)
- Due to security reasons, you can't use the QSECOFR profile in the SSH

It's important that the products are installed in this order:

1. IceBreak
 2. Portfolio
 3. IceCap
-

Installing IceBreak

Follow the below steps:

1. Download the newest version from this address:
<https://webfiles.system-method.com/download/sitemule/>
Follow the instructions in the install shield.
 2. If you have problems running the automatic installation or you can't connect to IBM i from the install shield, select "Manual installation" and follow the instructions.
-

Installing Portfolio

Follow the below steps:

1. Download the newest version from this address:
<https://webfiles.system-method.com/download/sitemule/>
 2. Follow the instructions in the install shield.
-

3. If you can't connect to IBM i from the install shield, select "Manual installation" and follow the instructions.

We strongly recommend installing the demonstration environment. The libraries CORPDATA and CORPPGM can be removed at any time by deleting them.

We also recommend that you also install Portfolio Client. The Client will provide the user with the feeling of a "real" program that can run on Windows, Mac, and Linux. You will find the install shields at this address: <https://webfiles.system-method.com/download/sitemule/>

Installing IceCap

Follow the below steps:

1. Download the newest version from this address:
<https://webfiles.system-method.com/download/sitemule/>
2. Follow the instructions in the install shield.
3. If you can't connect to IBM i from the install shield, select "Manual installation" and follow the instructions.

We strongly recommend installing the demonstration environment. The libraries CORPDATA and CORPPGM can be removed at any time by deleting them.

Uninstalling

To uninstall, please follow the below instructions:

IceBreak, the command QUSRSYS/DLTICEBRK PGMLIB(ICEBREAK) will:

1. Delete the library: ICEBREAK
2. Delete the IFS path: /www/icebreak
3. Delete the profiles: BLUEBOXUSR and WEBUSER

Portfolio, the command QUSRSYS/DLTPORTF PGMLIB(PORTF) DTALIB(PORTFDB) will:

1. Delete the libraries: PORTF and PORTFDB
2. Delete the IFS path: /www/iceapp/portf/

IceCap, the command QUSRSYS/DLTICECAP PGMLIB(ICECAP2) will:

1. Delete the library: ICECAP2
2. Delete the demo libraries: CORPPGM and CORPDATA
3. Delete the IFS path: /www/iceapp/icecap2/

Getting started

This chapter covers the basics of IceCap and introduce various aspects to help you understand how IceCap works. To fully grasp the capabilities of IceCap, Portfolio, and IceBreak, it is necessary to read the other chapters as well.

You don't need programming skills to install, set up, and use IceCap. However, if you have programming knowledge, you might find the examples in this chapter interesting due to the simplicity with which they solve complex tasks.

Installing

IceCap runs as an application within the Portfolio environment and is powered by an IceBreak server. If you haven't installed these three products yet, please follow the instructions in the chapter "Installation Guide".

You can access the IceBreak Administration by:

- Command line: `GO ICEBREAK`
- Browser: <http://myIBMi:7000>

Running Portfolio first time

To start Portfolio, open your browser and enter the following URL: `http://myIBMi:7700`

The Portfolio login screen will appear. You can use your IBM i™ user profile, but initially, you should use `QSECOFR` or another profile with `SPCAUT(*SECADM)`. This profile provides full access to the Portfolio System.

Portfolio logs you into the IBM i and then opens new applications in subsequent windows using that profile, similar to how Windows or Mac systems operate. This means you shouldn't be asked to log in again. Each window can display a 5250 session, a web application, simple HTML applications, links to the Internet, or other resources compatible with browser technology.

One of the key features is that each tab or window can communicate with others within the Portfolio session. We will cover interprocess communication in more detail later.

After the Login

Upon logging in, the main window appears with the following layout:

- **Navigation Panel** (top) - Select systems or arrange Portfolio
- **Menu Structure Panel** (left) - Find and start applications
- **Application Panel** (middle) - Where tabs and windows open
- **Help Panel** (right) - Provides help for the active application

Navigation

Users can select systems and arrange tabs, windows, and user settings in the top navigation panel.

Menu Structure Panel

The Navigator is divided into sections that can be expanded or collapsed. Each section contains subsequent menu options, defined in the administration or served by a customized Menu-Loader. More details can be found in the appendix "Customized Menu-Loader".

- **Search Feature** - Each menu section has an optional search feature to help users easily find desired applications.
- **Favorites** - Frequently used applications can be dragged into a separate Favorites section. Applications across the system and menu sections can be organized and arranged according to the user's preferences.
- **Click or Right-Click** - Clicking on an application opens a new tab or jumps to an active tab with the application. Right-clicking opens the same application multiple times in new tabs, allowing you to run multiple 5250 sessions side by side.

The Menu Structure Panel can be hidden by clicking the "<" at the top, providing more space for the Application Panel.

Application Panel

All applications run within the Application Panel in their own tab or window. To close an application, click the X in the upper right corner of the tab or window. Right-click on the tab to manage current and other tabs. If you have many tabs open, use the small arrows on either side of the tabs to scroll through them.

Help Panel

The Help Panel displays help content related to the active application. Users with administrative credentials can create and edit this content. This panel is deactivated by default.

Refer to the chapter “Portfolio” for more detailed information.

Running IceCap first time

Portfolio is installed with at least with these 4 applications called IceCap Flat, IceCap Flex, IceCap Flow, IceCap Flip in the section “IceCap” representing the various presentation modes.

Get familiar with the modes

By clicking on “IceCap Flat” you will automatically be logged on IBM i. You properly recognize the screen as the same screen that you would see if you were logged on a 5250 terminal in the IBM Access Client Solution. If you ‘SIGNOFF’ you would also recognize the traditional login screen.

Depending on the setup of your ‘Initial Program’ or ‘Initial Menu’ on your IBM i User Profile you might already be seeing the menu system in your ERP solution. If not, please call the program or menu you normally use on IBM i.

Take tour through the different types of applications your ERP solution provides. You should experience a very close similarity to the traditional 5250 screens you are used to see. To get familiar with the other modes please repeat the “tour” for the other three applications IceCap Flex, Flow, and Flip.

The modes are just predefined presentations that can be customized to fit your ERP solution better. It's not unusually that it is necessary to make some adjustment, if so refer to the chapter in the "IceCap" section of the manual.

Tour the Demo Environment

You should also take a tour through the Demo Environment as well to experience the variation of the presentation modes and the many extra functions and visual effect we have added with IceCap. This tour is also important if you experienced some deviations on your tour through your ERP Solution in thye different modes

Accessing a 5250 program from Portfolio

To understand how programs or commands are executed in Portfolio, let's compare the process to the 5250 environment. In a traditional 5250 setup, you would call a program with a command like:

```
CALL CORPPGM/EMPLOYC
```

In **Portfolio**, the process is similar, but instead of directly calling the program, it is passed as a function parameter within the IceCap application (using `.aspx`). This dynamically converts the 5250 program into a web application. For example, you would use the following URL to call the program:

```
IceCap.aspx?func=CALL CORPPGM/EMPLOYC
```

You can also pass additional parameters as required, just like in a 5250 environment. For instance, to run an IBM i command through IceCap, you would use:

```
IceCap.aspx?func=WRKSPLF SELECT(*ALL)
```

To invoke a 5250 program or IBM i command directly from a browser, which will open within Portfolio like any other menu item, you can use the following URL format:

```
http://myIBMi:7700/#IceCap.view.Main-{}-{"func":"WRKSPLF SELECT(*ALL)"} }
```

This will launch **Portfolio** and validate your credentials using the standard login procedure. If you

want to restrict IceCap to a single tab (without launching the full Portfolio environment), use this format:

```
http://myIBMi:5250/?options={"func":"WRKSPLF SELECT(*ALL)"}&single=true
```

Please note that the port number (e.g., 5250) in the examples above refers to the default for a standalone IceCap installation. If IceCap is not installed as a separate instance, this example may not work as shown.

This feature is particularly useful for embedding 5250 programs or commands into other platforms, such as within an embedded browser in a Windows application. It is especially beneficial if the Windows application supports inserting dynamic variables into the URL prior to execution. For more details on this integration, refer to the "Integration" chapter.

Auto Login and Layout Customization

An alternative approach is to open **Portfolio** and bypass the login screen by specifying the username and password in the URL, while also suppressing most Portfolio components by choosing a layout number (e.g., layout 31). The URL would look like this:

```
http://myIBMi:7700/?layout=31&username=pno&password=john#IceCap.view.Main-{}-{"func":"WRKSPLF SELECT(*ALL)"}&single=true
```

Before using this method, be aware of the security implications of sending credentials through a URL. Additionally, this feature must be confirmed in the configuration manifest file located at `/www/iceapp/<portfolio>/manifest.js`. Make sure that the `AutoLogon` feature is enabled by adjusting the following parameters:

```
Object.assign(Ext.manifest, {
  singleSession: true,
  showHelp: false,
  stateful: false,
  showErrors: true,
  showBottom: false,
  switchType: false,
  switchTheme: true,
  changePassword: true,
  showNotifications: true,
  autoLogon: true,           <-- confirm AutoLogon
  start: 1,
```

```
open: 1,  
frame: 1,  
...
```

For more details about layouts, refer to the "Linking to Applications in Portfolio" chapter.

Setting up Portfolio

Portfolio is installed with a predefined menu structure:

- **Application** - IceCap and optional Demo Environment
- **IBM i** - Complete set of command menus on IBM i
- **System** - Menus, users, roles, permissions, and other administrative utilities

You can maintain the menu structure by clicking "SYSTEM" in the top navigation panel. It is recommended to install the Demo Environment to explore the setup and use it as a template for your own system. Remember, you can right-click to copy items.

Instead of creating a copy of the existing menu in your ERP Solution, you should access the menu directly from Portfolio with a "Menu-Loader".

Access your ERP Solution from Portfolio

In Portfolio, a **Menu-Loader** is a RESTful web service component that creates a dynamic menu structure in JSON format. It is designed to integrate and display existing 5250 menu systems or custom menus within the Portfolio web interface. The Menu-Loader retrieves menu items, verifies user access rights, and provides a well-organized navigation system.

By dynamically generating easy-to-navigate menus, the Menu-Loader allows users to seamlessly interact with their familiar ERP systems through a modern browser interface.

Typically, a Menu-Loader loads items from a Db2 table and checks access rights against user credentials. A template program can be found in the appendix "Customized Menu-Loader."

The "IBM i" menu structure is dynamically generated and presents a vast menu structure in multiple levels as an example on what a Menu-Loader can provide.

Setting up IceCap

The initial setup of IceCap is completed once you have installed the products and familiarized yourself with Portfolio, IceCap, the available presentation modes, and how these tools integrate with your ERP solution.

Customization Features in IceCap

IceCap offers several customization options to help tailor the solution to fit your ERP system and streamline workflows. These five key features: **Themes**, **Modes**, **Keywords**, **Tags**, and **Plugins** - can be used independently or in combination. While users can easily switch between Themes and Modes, Keywords and Tags require technical adjustments. Plugins provide a flexible development framework for RPG Free developers to add functionality to the 5250 datastream without altering the original programs.

Themes

IceCap allows you to adjust its appearance using a variety of predefined themes. These themes enable you to modify the color schemes and overall interface to align with your organization's branding or individual user preferences.

Applying a theme is simple—users can select the preferred theme from the available options in user settings. Multiple color themes are available and can be switched without affecting the functionality of the application. Themes not only enhance the user experience but also provide a professional and cohesive look for modernized IBM i applications.

Modes

IceCap offers several presentation modes that transform how 5250 screens are displayed, providing different levels of modernization:

- **Flat:** A basic 5250 web emulation.
- **Flex:** A modernized appearance with proportional fonts.
- **Flow:** Converts subfiles into tables and organizes function keys into menus.
- **Flip:** Transforms the entire interface into a Windows-like environment.

These modes are composed of more than 40 available keywords. You can easily modify the existing modes or create new ones by customizing the keywords. For details, refer to the “Presentation Keywords” section in the Configuration chapter.

Keywords

Keywords are powerful configuration options that allow you to control the behavior and appearance of applications without altering the underlying code. By defining keywords in the configuration file, you can adjust elements such as button visibility, screen layout, function keys, and field behavior.

Keywords offer flexibility and precision, enabling you to customize the interface and functionality of IceCap to meet specific business needs. Before using keywords, review the “Presentation Keywords” section in the Configuration chapter for a complete list of available options.

Tags

Tags in IceCap enhance the user interface by transforming how specific fields or screen elements are displayed. By identifying certain text or fields in the 5250 datastream, tags can add interactive elements like drop-down lists, checkboxes, date pickers, and improved field validation.

Tags are easy to configure through IceCap’s settings. Once applied, they automatically update the interface without requiring changes to the underlying program logic. Tags offer a simple way to modernize the user experience and streamline workflows in IBM i applications without the need for programming.

Plugins

Plugins provide a flexible method for extending IceCap’s functionality. They allow you to add new features, integrate third-party services, or modify existing workflows without altering the core logic of the application. Plugins can be used on both the client-side (e.g., enhancing the user interface) and server-side (e.g., automating data processes).

To implement a Plugin, register it in the IceCap configuration file and define its behavior. Client-side Plugins might include interactive elements like maps or calendars, while server-side Plugins can automate complex business logic or enhance data processing. Plugins provide a powerful way to tailor IceCap to your specific business requirements, improving overall user experience and workflow efficiency.

Coding with IceCap

In IceCap, coding is only required when creating new Menu Loaders or developing Plugins. Both IceCap and Portfolio offer various template programs that assist in coding tasks. These templates can be copied, customized, and compiled, allowing users to complete tasks without deep technical expertise. The templates make it possible for users with limited coding experience to develop functionality with support and guidance.

Working with Plugins

Plugins are a central feature in IceCap that allow developers to extend and enhance the core program logic. They serve as breakpoints where IceCap either diverts from the original program flow or integrates additional client- or server-side components.

- **Server-Side Plugins** typically focus on adding or retrieving data.
- **Client-Side Plugins** usually manage visual components or provide interactivity.

These two types often work together: for example, the server-side plugin retrieves the data, and the client-side plugin displays it.

Practical Example: Plugin for Data Validation

Consider the scenario where users must manually enter a department code without a search (F4=Prompt) or validation option. A plugin can be developed to improve this task.

In this example, a plugin is designed to add a combo box that contains a SQL query result to the 5250 screen. The combo box allows users to validate their entries, search department codes, and conduct searches based on department names. The plugin is triggered every time a 5250 screen is displayed, accessing the screen's display buffer to manipulate input fields.

Server-side plugins in IceCap are developed using RPG. To implement a plugin, you will need to create and compile an ILE (Integrated Language Environment) program. The process involves using APIs provided by IceCap to hook into the Virtual Terminal session and manipulate 5250 screens.

RPG Code Example

We want to limit the use of the Plugin to screens with the field label "Department" in line 11 starting position 4 and an opened input field for the department code located on the right side (East). We set up the SQL statement in the string `sqlStmt` and inserted it into the JSON structure `pMeta`. We

specify that we want the client side to use the 'xtype' DDSQL (Drop Down SQL) to present the data and enable the various search methods and properties like: width, maxLength, valueField, splitSearch, and displayField.

```
<%
ctl-opt copyright('System & Method (C), 2021');
ctl-opt decEdit('0,') datEdit(*YMD.) main(main);
ctl-opt text('ICECAP: Simple emulator');
ctl-opt bndDir('ICECAPTERM':'ICECAPEMUL':'EXTUTILITY');

/Include qrpgleref,iceCapTerm
/Include qrpgleref,iceCapEmul
/include qrpgleref,iceCapTool
/include qasphdr,iceUtility
/include qasphdr,jsonparser
/include qasphdr,xmlparser
/include qasphdr,extUtility

/* -----
Main Line:
----- */
dcl-proc main;

    dcl-pi *n ;
           vts           likeds(vtsDS);
    end-pi ;

    dcl-s pVts           pointer;
    dcl-s pMeta           pointer;
    dcl-s pSql           pointer;
    dcl-ds vtLocation     likeds(vtLocationDS);
    dcl-s sqlstmt         varchar(8192);

    setContentType('application/json; charset=utf-8');
    setEncodingType('*JSON');
    json_setDelimiters('/\@[ ] .{}'"$');

    // GET THE Virtual Terminal
    pVts = %addr(vts);

    // Get input Location
    vtLocation = vtLocate(
        pVts
        :VT_INPUT
        :vtLinCol(4:11) // Line : column
        :VT_EAST
        :'Department'
    );
```

```
// If input is located
if vtLocation > *loval;

// CREATE THE SQL
sqlStmt = (`
    with rows as (
        select
            DEPTNO concat ' - ' concat DEPTNAME as "text",
            DEPTNO as "value"
        from CORPDATA.DEPARTMENT
    )
`);

sqlStmt += (`
    select *
    from rows
    :where
    order by "text"
    limit :limit offset :start
`);

// Make sure tht we translate to DKK for compa
sqlStmt = xlateStr(sqlStmt: 277:0);

// GET THE pointer to the sql
pSql = json_parseString(extSql('{}' :sqlStmt));

// CREATE JSON WE SEND BACK
pMeta = json_newObject();

// Set properties
json_setStr(pMeta:'xtype':'DDSQL');
json_setint(pMeta : 'width': 200);
json_setint(pMeta : 'maxLength': 90);
json_setstr(pMeta : 'sqlStmt': sqlStmt);
json_setStr(pMeta : 'valueField': 'value');
json_setbool(pMeta : 'splitSearch': *on);
json_setStr(pMeta : 'displayField': 'text');
json_copyValue(pMeta: 'sqlVoucher': pSql: 'sqlVoucher');

// Set the data on the input
vtSetMetaData(
    pVts
    :vtLocation
    :JSON_ASJSONTEXT16M(pMeta)
);

json_delete(pMeta);
```

```
endif;  
  
return;  
  
end-proc;
```

Understanding the Code

The initial lines, known as **h-specs**, specify the use of APIs supported by IceCap and ensure integration with the activation group that IceCap provides. The **'include'** statement imports the necessary prototypes and data types, such as the Virtual Terminal Session (VTS) structure. It's important to note that all IceCap APIs require a pointer to the VTS. In the first part of the code, this is achieved by using the **%addr()** built-in function to store a pointer to the VTS.

In addition, all data structures are templates, so you must define your own structures using the **'likes'** keyword. All IceCap APIs are prefixed with **'vt'**, and the emulator APIs are prefixed with **'eml'**. For example:

```
loc = vtLocate(pVts: VT_INPUT: FromLin: FromCol: VT_EAST: 'Department');
```

The **vtLocate** function identifies the position (line and column) of the first input field located to the right (east) of the field label (keyword) "Department Name." The search begins from the specified starting column (**FromCol**) and line (**FromLin**). If the keyword is not found, it returns a location of 0/0.

Once the field is located, additional features can be applied, such as assigning an **'xtype'** for a drop-down list of departments. All the xtypes are described in the "Web Components ('xtype')" appendix.

```
vtSetMetaData(pVts: loc: '{xtype:"DDSQL"}');
```

The **vtSetMetaData** function supplies the emulator with the required details to display an input field at the specified line and column as a different type. In this example, it will apply to all screens where input fields are labeled "Department Name."

This method is highly efficient, allowing changes to be implemented across multiple programs and systems with minimal configuration. To restrict these changes to particular systems or screens, you can detect the panel name and take appropriate actions.

```
if vtSaaTitleContains(pVts: 'Change employee');  
    vtSetMetaData(pVts: loc: '{xtype:"DDSQL"}');  
endIf;
```

As per the IBM Systems Application Architecture (SAA) guidelines, the screen title is typically centered on the first line. If an application adheres to SAA standards, you can use `vtSaaTitleContains()` to detect the panel title.

However, many applications follow their own conventions. In such cases, you can identify the application name and the specific screen displayed by using a combination of API calls, such as:

```
if vtGetScreen(pVts: Lin1: Col1: Len1) = 'EMPLOYDB' and vtGetScreen(pVts: Lin2: Col2: Len2)  
= 'Corp. Data';  
...  
...
```

In some cases, you may want to prevent a specific screen from being displayed to the user or merge multiple screens into one. To achieve this, you'll need to navigate the application using function keys such as `ENTER`, `Page-Up`, `F3`, etc. Additionally, there may be situations where you need to input data into the application programmatically. The relevant API for this process is as follows:

```
vtSetField(pVts: lin: col: 'MyValue');  
vtPressKey(pVts: vtENTER);
```

For detailed API descriptions, refer to the "Virtual Terminal APIs" appendix.

Interprocess Communication

IceCap enables communication between 5250 sessions within the Portfolio environment. This functionality enables direct access to the shared Portfolio session from the active 5250 sessions, allowing the management of session variables, execution of web services, creation of Internal Large Objects (ILOBs), and other advanced operations.

To utilize this feature, the IceBreak bind directory must be included in your program. It's crucial to note that these APIs will not affect programs running in a standard 5250 emulator like Access Client Solutions (ACS).

Practical Example

In traditional workflows, a user may begin by searching for a key value, such as a customer or order number, in one 5250 program. Afterward, the user must open another program and manually re-enter the same key values to continue. This process is often tedious and time-consuming, leading users to write down these key values for future reference.

One approach to resolving this issue would be to rewrite all 5250 programs to accept parameters when called from the search program, but this approach is cumbersome. Instead, IceCap's interprocess communication provides a more efficient solution. Session variables allow Portfolio to store key values from one session and pass them internally to subsequent programs, even across different 5250 sessions.

Program Example

In the search program, the employee number can be stored in the Portfolio session using the `sesSetVarNum` function:

```
H BNDDIR('ICEBREAK')
/include qasphdr,httpxserv
...
C                EXFMT  MAIN // Original code with EMPNO
C                CALLP  sesSetVarNum('employeeNo' : EMPNO)
...
```

In the subsequent program, the employee number is retrieved from the shared Portfolio session using `sesGetVarNum`:

```
H BNDDIR('ICEBREAK')
/include qasphdr,httpxserv
...
C                eval   EMPNO = sesGetVarNum('employeeNo')
C                EXSR   CHAINEMP // Original code now using the EMPNO
C                EXFMT  MAIN      // Original code
...
```

This enhancement enables legacy 5250 programs to communicate seamlessly within the Portfolio session, eliminating the need for manual re-entry of data. Although in the past, a local data area

(*LDA) might have been used for such communication, it is less flexible due to limited data capacity and the absence of named keys.

Greater Flexibility

Instead of modifying every 5250 program, you can integrate the **SET** and **GET** functions into the Emulation component, which intercepts the 5250 screens and implements the interprocess communication feature without altering the original programs. This method significantly improves the workflow efficiency between programs with minimal effort.

Interprocess Communication is just one of the many features described in this manual that will enhance workflows in the 5250 environment.

We hope the chapter has helped you **get started** with IceCap and Portfolio.

IceCap

Presentation

IceCap can be tailored to meet the needs of various customers. Configuration settings are managed in the `<config>` section of the `IceCap_config.xml` file, with most settings controlled by specific keywords.

Additional configurations can be programmed in the component templates for Emulation and Interpretation. For more information, refer to the chapter "Plugins."

Before altering component settings, consider using Tags. Tags are more intuitive and do not require programming. For more details, see the chapter "Tags."

Predefined Modes

To assist customers in getting started, IceCap includes several standard configurations. Currently, there are four modes available:

IceCap Flat	Provides a flat, one-to-one 5250 emulation on the web. The screen title is used as the tab/window title, and function keys (F-keys) are reformatted as buttons.
IceCap Flex	Displays screens with a proportional font for a more flexible appearance. All F-keys, including those behind F24=More keys, are shown.
IceCap Flow	Converts subfiles into tables, allowing options to be accessed via right-click selections. All function keys (F-keys) are organized into a dedicated function menu, with essential F-keys displayed on the screen.
IceCap Flip	Transforms the entire screen into a Windows-like interface, displaying some F-keys as tabs and others as buttons. The characteristics option and F-key numbers are removed.

The appendix "Keywords" provides an overview of all keywords and documents the default settings

for different presentation modes. The general default setting is IceCap Flat, found in the `<default>` section of `IceCap_config.xml`. The other modes are `<flex>`, `<flow>`, and `<flip>`.

Combining Modes

When setting up applications in the menu structure, the mode (or config) is selected per application, allowing for the combination of modes within the same system. Often, a single mode is set for the entire system to ensure a unified presentation for users.

Defining your own Mode

Different 5250 systems may have unique visual and functional requirements that necessitate individual settings.

To create a custom presentation mode, follow these steps:

1. Open `IceCap_config.xml` in an editor.
2. Copy the structure that most closely matches your desired presentation.
3. Assign a suitable name to the copied structure, such as `<sales>` or `<warehouse>`.
4. Modify and add keywords as needed to customize the presentation mode.
5. Save the changes to `IceCap_config.xml`.
6. Insert the new mode as the config name in the menu structure of the affected programs.
7. Restart the IceCap server in IceBreak to apply the changes.

Below is the configuration written in XML to copy if you want to create a custom presentation mode similar to IceCap Flip:

```
<flip>
  <defaults
    trace="no"
    tracePath="/www/iceapp/icecqa/trace/"
    logClientActions="no"

    showOptionColumn="no"
    showOptionNo="no"
    showKeysAsTabs="yes"
    showKeysAsPredefined="F6,F9"
```

```
showKeysFxx="no"
showKeysToolTip="yes"
showStatusBar="no"
showPageUpDown="yes"
showLabelUnderlined="yes"
showEnterKey="yes"
enterKeyText="OK"
useInterpretation="yes"
useTags="yes"
useSubfileTransformation="yes"
/>
<breakOut
  navigatorProgram="*LIBL/*NONE"
  emulatorProgram="*LIBL/EMLFLW"
  interpretationProgram="*LIBL/INTTAG"
  pcoProgram="*LIBL/PCOSERVER"
  i18nXlateProgram="*LIBL/*NONE"
  xxx_deviceNameProgram="*LIBL/*NONE"
  xxx_varyoffExitProgram="*LIBL/*NONE"
  clientXtypesPath="/menu/scripts/xtypes.js"
  clientScriptPath="/menu/scripts/breakouts.js"
  clientCSSPath="/menu/styles/changeFontSize.css"
/>
</flip>
```

For more detailed information, refer to the chapter "Configuration."

What does IceCap change?

In essence, IceCap only modifies the presentation of traditional 5250 black-and-green screen programs. It does not alter the core functionality of the **IBM i** operating system, such as subsystems, jobs, batch jobs, queues, libraries, objects, commands, programs, users, authority, files, spooled files, journals, logs, communication, or configuration. Similarly, IceCap does not change any functionalities within the **ERP** or other systems running on **IBM i**.

However, IceCap enhances the system by offering the capability to modify and streamline workflows on **IBM i**. This allows for more efficient processes while keeping the underlying systems intact.

Here are some key examples of how IceCap can enhance operations:

- **Send Notifications:** Trigger alerts when a job ends, when data is changed, or when a process requires attention.

- **User Interface Modernization:** Transform the traditional user interface into a modern and intuitive experience.
- **System Integration:** Integrate with internal or external systems and services on other platforms, improving connectivity.
- **Faster Navigation and Multitasking:** Enhance efficiency by speeding up navigation and enabling multitasking capabilities.
- **Field Labeling and Validation:** Easily adjust field labels and validations to ensure data clarity and integrity.
- **Workflow Adaptation:** Adjust workflows to meet current and future business needs.
- **Automation of Repetitive Tasks:** Automate repetitive data-entry tasks, reducing errors and increasing processing speed.

Tags

Tags in IceCap are powerful tools that allow you to customize and enhance the user interface of your IBM i applications without requiring extensive programming skills. They enable you to add new functionalities, improve usability, and optimize workflows by modifying how specific fields or screens are presented and interacted with in a web-based environment.

What Are Tags?

Tags are text strings that identify specific elements within the 5250 datastream, such as field labels, codes following fields, screen titles, program IDs, or any text on the screen. When IceCap processes the datastream through its Emulator component, it actively searches for these Tags. Upon detecting a Tag, the system checks for an input field adjacent to it—initially on the left side. If an input field is found, IceCap enriches it using the JavaScript configuration associated with the Tag. If no input field is present on the left, the system searches on the right side and applies the same procedure.

In this tagging process, priority is given to codes (e.g., (YMD), F4) over field labels (e.g., Department) because codes are generally placed after the input field, whereas labels are positioned before it. This prioritization can be customized by configuring the Tag to search exclusively on either the left or right side.

By using Tags, you can:

- **Add visual aids** such as combo boxes, checkboxes, radio buttons, and date pickers.
- **Improve validation** and field navigation.
- **Automate data entry** and enhance workflows.

Once a field or screen is tagged, IceCap automatically changes how this element is presented and functions in the web client. Using Tags is a user-friendly alternative to making program changes in the Emulation, Interpretation, and Navigation components.

If field labels need to be changed or translated, this can be accomplished using the Translation feature, which also supports multilingual labels. Refer to chapter “Translation” for more detailed information.

Accessing and Managing Tags

To access the Tag definitions, navigate to the system settings and select IceCap settings. Within this section, you will find the "Tags Definition" area, where a grid displays all available Tags, including those predefined during the installation of IceCap. These Tags are a combination of sample configurations and actual Tags used in the demo environment. Familiarizing yourself with the demo environment can be beneficial before making modifications.

Viewing and Organizing Tags

For easier management, Tags can be sorted by type. To sort, click on the "Type" column and drag the "Type" box to the left in the filter area. This sorting action will categorize the Tags, making it easier to locate different types, such as; Calendar, Checkbox, Link, and Multiple Choice. This provides a structured view of Tags, simplifying the process of identifying and organizing them.

Tag Definitions

Each Tag in IceCap is identified by a unique ID and consists of several components:

- **Tag ID:** The unique identifier corresponding to the text or code that IceCap scans for within the datastream.
- **Type:** Categorizes the Tag for organizational purposes.
- **Description:** An internal field describing the Tag's purpose and functionality.
- **Config:** Contains the JavaScript configuration dictating how the component operates. For example, a Tag of type **DDSQL** (DropDown SQL) might execute an SQL command when the field is entered.
- **ToolTip:** Text displayed when a user hovers over the field, which can be formatted using HTML.
- **EmptyText** (Placeholder): Sample input format displayed within the field to guide the user.
- **Location:** Specifies whether the input field is located on the left, right, or both sides of the Tag.
- **Remove:** Determines whether the Tag is removed from the datastream after processing.
- **Protected:** Ensures the Tag definition is not overwritten during system upgrades if it has been modified.

- **Status:** Indicates whether the Tag is active and applied across the system.

When saving the modification will take effect immediately.

Tags Definition

Department - Tags

Content

Info

Tag ID:

Department

Type:

Search Database

Description:

CORPDATA - Project, Department No.

Config:

```

1  {
2      "type": "DDSQL",
3      "library": "CORPDATA",
4      "fileName": "DEPARTMENT",
5      "flex": 30,
6      "width": 400,
7      "textColumns": [
8          "DEPTNO",
9          "DEPTNAME"
10     ],
11     "valueColumn": "DEPTNO"
12 }

```

Mouse-Over:

Search by department number or name - or type [Cursor Down] to select from a complete list of departments.

Example Text:

Indicate the department to which the project is related.

Remove:

☒ Remove Tag after use.

Protected:

☒ Protected from product upgrades.

Status :

☒ Active

Identifying Tags

When the 5250 datastream is processed through IceCap's Emulator component, every piece of text is scanned for Tags. The system's procedure for identifying and processing Tags is as follows:

1. **Scan for Tags:** The emulator scans the text for any defined Tags.

2. Search for Adjacent Input Fields:

- Left Side: Checks for an input field on the left side of the Tag.
- Right Side: If no input field is found on the left, it searches on the right side.

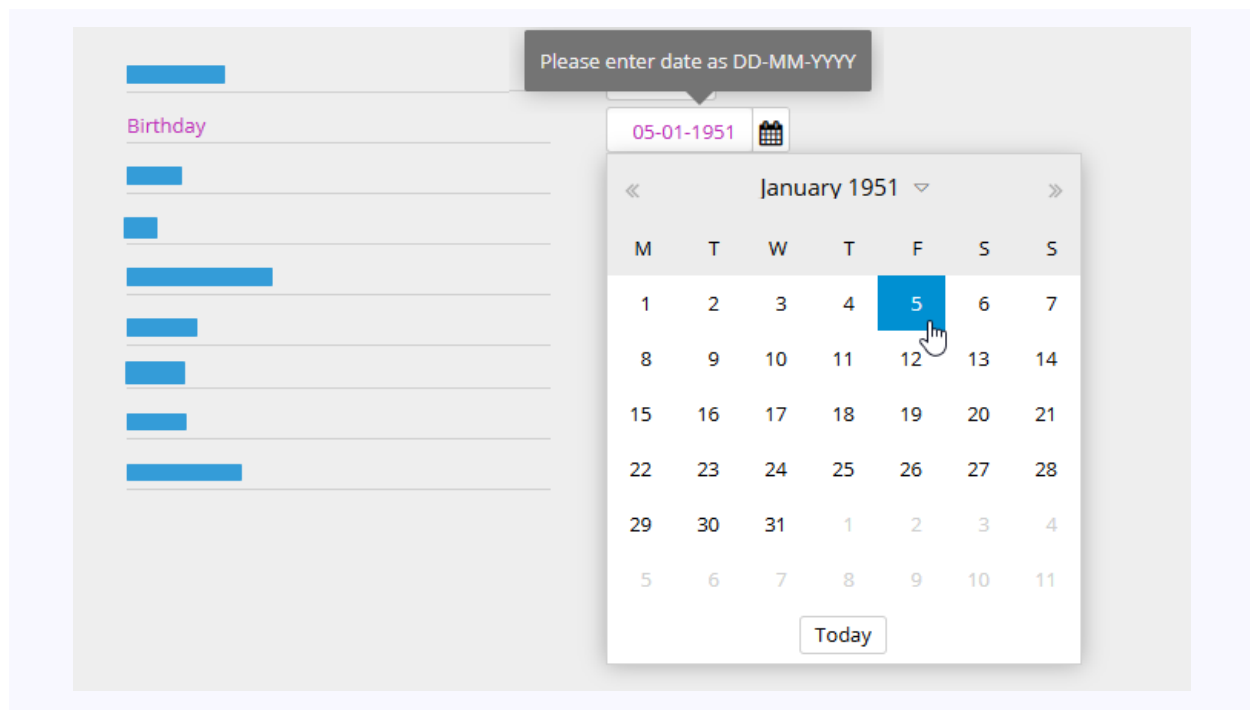
3. Apply Configuration: If an input field is found, the system enriches it using the associated JavaScript configuration from the Tag definition.

4. Prioritization: Codes are prioritized over field labels due to their typical placement relative to input fields. This behavior can be adjusted in the Tag's configuration settings.

Practical Examples

Calendar Tag

This example shows how a date field is presented when a calendar Tag have been added:

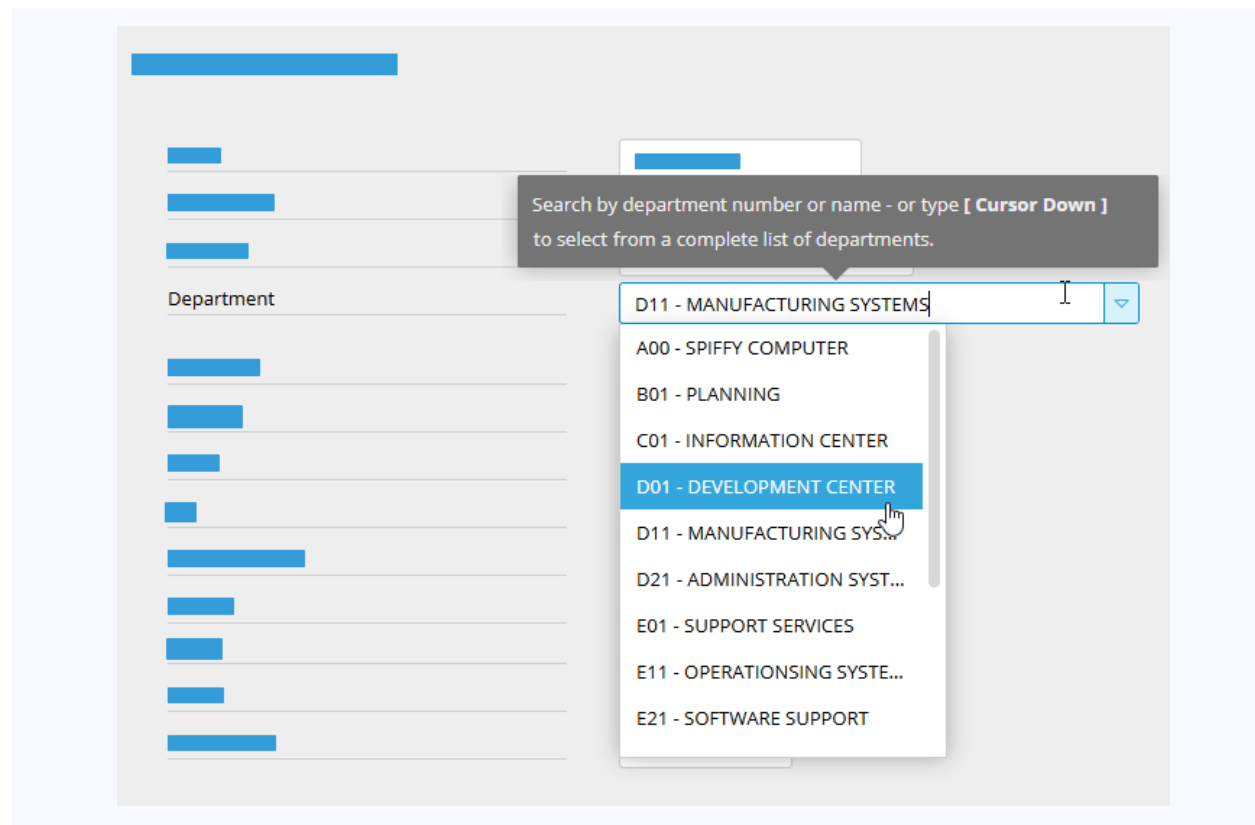


- **Tag ID:** (DMYY)
- **Type:** Calendar
- **Description:** Used to present a date field formatted as day-month-year (DD-MM-YYYY) with an integrated calendar picker.

- **Config:** Specifies the date format and enables the calendar functionality.
- **Usage:** When the Tag (**DMYY**) is detected adjacent to a date input field, IceCap replaces the standard input with a date picker, enhancing usability and ensuring valid date entries.
- **ToolTip:** Provides additional guidance on date entry.
- **EmptyText:** Displays a sample date format (e.g., **DD-MM-YYYY**) within the field.

Combobox Tag

The example show how a database lookup xxxx



- **Tag ID:** **Department**
- **Type:** Combobox
- **Description:** Converts the department input field into a dropdown list populated with department names.
- **Config:** Executes an SQL query on the **CORPDATA/DEPARTMENT** table (in the Demo Environment) to retrieve department names.

- **Usage:** Enhances data entry by allowing users to select from existing departments rather than typing them manually.
 - **ToolTip:** May include HTML-formatted guidance on selecting a department.
 - **EmptyText:** Provides sample input or instructions within the field.
 - **Remove Tag After Use:** Should be disabled to prevent the field label from being removed after application.
-

Web Components

IceCap includes a variety of UI web components, known as 'xtypes', that can be used in conjunction with Tags to enrich and streamline the 5250 interface. These components address common requirements when modernizing the interface and are utilized extensively in the demo environment. Some of the key components include:

- Action Field
- CheckBox
- ComboBox
- Calendar
- DropDownSQL
- Function Field
- Map
- Number Field
- Radio Buttons
- Text Field
- Web Search

For detailed technical information on these components, refer to Appendix E - "Web Components ('xtype')." [View Appendix E](#)

Benefits of Using Tags

Using Tags in IceCap enhances usability by allowing the addition of modern interactive elements like combo boxes and date pickers, which improve the user experience and facilitate faster, more

accurate data entry. Tags can be implemented without extensive programming knowledge, making it accessible for non-technical users to customize the interface to their specific needs. This flexibility and customization capability lead to a more user-friendly and efficient working environment, ultimately boosting productivity and reducing errors.

Plugins

Plugins are a fundamental aspect of IceCap that provide an effective way to extend and customize the functionality of your applications. They enable developers to introduce new features and integrations, enhancing the user experience and streamlining workflows.

What are Plugins?

Plugins in IceCap allow for the incorporation of additional client-side or server-side components into the existing application framework. These components can either modify the behavior of the application or add new functionalities, without altering the core logic of the original program. Plugins are particularly useful for integrating third-party services or implementing specific business logic that extends the capabilities of the base application.

Types of Plugins

There are two main types of Plugins in IceCap:

1. **Client-side Plugins** - These are implemented to enhance the user interface and user interactions within the browser. For example, integrating Google Maps to display customer locations or adding a date picker for easier date selection.
 2. **Server-side Plugins** - These Plugins operate on the server and can modify the data or workflow before it is sent to the client. For instance, a server-side Plugin could automate data validation or enrich data fields before displaying them on the user's screen.
-

Implementing Plugins

To implement a plugin in IceCap, follow these steps:

1. **Define the Plugin**

Based on the plugin's purpose, assign it to one of the relevant components: Emulation, Interpretation, Translation, Navigation, or PC-Organisation (STRPCO).

Emulation-Related Components: If the plugin needs to manage or modify the 5250 datastream presentation on the web client, place it in Emulation, Interpretation, Translation, or PC-Organisation. This includes tasks such as transforming screen appearance, managing JSON structures for enhanced UI, or customizing function key displays. These components are ideal for visual and interactive enhancements that improve individual screen engagement without altering business logic.

Navigation Component: Use the Navigation component when the plugin aims to streamline workflows across multiple programs. This is particularly useful for chaining programs, adding business logic, or automating screen transitions. Plugins in Navigation enable seamless workflows by intercepting the data stream to combine programs, automate data entry, and reduce redundant actions.

2. Develop the Plugin

Use the appropriate technology stack for plugin development. Client-side plugins typically involve JavaScript and are built with the ExtJS framework, while server-side plugins can be developed in RPG Free or other IBM i-compatible server-side languages.

3. Configure the Plugin

Register plugins in the IceCap configuration XML file, usually found at /www/iceapp/IceCap2/icecap_config.xml. Within the `<breakouts>` section, specify the component's library and program name. Restart the IceCap server after making changes for the configuration to take effect. See XML example below:

```
<corpflow>
  <defaults
    trace="no"
    tracePath="/www/iceapp/icecqa/trace/"
    logClientActions="no"

    showOptionSeparator="- "
    showKeysToolTip="yes"
    showKeySeparator=" "
    showKeysMenu="yes"
    showLabelUnderlined="yes"
    useInterpretation="yes"
```

```
        useTags="yes"
        useSubfileTransformation="yes"
    />
    <breakOut
        navigatorProgram="*LIBL/NAVCAPCORP"
        emulatorProgram="*LIBL/EMLFLWCORP"
        interpretationProgram="*LIBL/INTTAG"
        pcoProgram="*LIBL/PCOSERVER"
        i18nXlateProgram="*LIBL/*NONE"
        xxx_deviceNameProgram="*LIBL/*NONE"
        xxx_varyoffExitProgram="*LIBL/*NONE"
    />
</corpflow>
```

Example of a various Plugins

- **Emulation Program** - [\[Text under construction\]](#)
 - **Interpretation Program** - [\[Text under construction\]](#)
 - **Navigation Program** - [\[Text under construction\]](#)
-

Benefits of using Plugins

- **Enhanced Functionality** - Plugins allow for the integration of advanced features and third-party services, enhancing the capabilities of the base application.
- **Customization** - They provide a way to tailor the application to specific business needs without modifying the core codebase.
- **Improved User Experience** - Client-side Plugins can significantly improve the user interface and interaction, making the application more intuitive and efficient.
- **Seamless Integration** - Server-side Plugins can automate processes and integrate smoothly with existing workflows, reducing manual intervention and errors.

Plugins are a powerful tool in IceCap, enabling users to customize and enhance their applications to better meet their needs and improve overall productivity. By leveraging both client-side and

server-side Plugins, businesses can achieve a high level of flexibility and functionality in their software solutions.

Refer to the section "Interpretation" in Appendix Z for detailed code examples that illustrate how to develop a program aimed at enriching keyword values when executing IBM i commands.

Charts and Documents

IceCap provides enhanced functionality by adding extra tabs to the 5250 screen, offering additional workspace for related information relevant to the displayed data. IceCap supports various data formats, including PDF documents, HTML files, images, and visual data representations in charts and diagrams.

IceCap improves the visualization and organization of data within IBM i applications. By integrating charts such as pie charts, bar graphs, and line charts, users can analyze complex data more easily. This enhanced visualization aids in better data interpretation and decision-making.

The addition of extra tabs allows users to access supplementary information without navigating away from the main screen. This feature is beneficial when displaying related data, documents, or tools, ultimately improving user experience and boosting productivity.

The combination of charts and extra tabs transforms the interface into a more interactive and informative environment, promoting efficient and effective user workflows.

Setting Up Charts and Extra Tabs

Consider the following example: a user needs to compare salary composition (salary, bonus, and commission) for employees and review their job application forms. They want to visually compare the salary breakdown using a pie chart displayed alongside employee information on the 5250 screen. Additionally, they need to view the job application form in a separate tab and occasionally search for the employee's name on the internet.

Here's how you can set up this environment in IceCap:

1. **Identify the 5250 Screen:** The employee screen, which displays or edits employee information, is identified by screen titles like 'Display Employee' or 'Change Employee.' Once identified, a procedure sets up additional panels to support extra tabs.
2. **Add Three Extra Tabs:** Using a procedure, IceCap creates three tabs:
 - One tab contains the main 5250 screen alongside a pie chart.
 - The second tab displays the result of an internet search for the employee's name.
 - The third tab showcases a PDF document, such as the employee's job application form.

3. **Present a Pie Chart:** A pie chart is created using the employee's salary data (salary, bonus, and commission). The chart is placed beside the 5250 screen, allowing users to view both the data and its graphical representation.
4. **Show Internet Search Results:** The employee's name and surname are used to compose a search string. An iframe is embedded within the second tab to display the search results from the internet.
5. **Browse a Related PDF Document:** The third tab provides the ability to view a related PDF document, such as the employee's job application form. This document can be loaded dynamically based on the current employee's data.

Example

1. Recognizing the Employee Screen

The employee detail screen is identified by titles such as 'Display Employee' or 'Change Employee.' Once this screen is detected, a procedure is invoked to set up additional panels. The following example uses the `vtSaaTitleContains` command to recognize the screen:

```
// Not grid (Subfile)
if (pGrid = *NULL);
    pParams = json_newObject();
    json_setint(pParams : 'startLine' :1);
    json_setint(pParams : 'endLine' :keyStartLine - 1);
    json_setint(pParams : 'colorTheme' :colorTheme);

    pOutput = icecapCreateFlexLayout(pVts: pParams);
    json_delete(pParams);

    // Work With Employee
    // Add Additional 'tabs' for Internet Search and Job application to screen.
    if (vtSaaTitleContains(pVts : 'Display employee')
    or  vtSaaTitleContains(pVts : 'Change employee'));

        // Create additional "Tab Panels"
        pPanel = create_panels(pVts:pOutput);

        return pPanel;

    endif;
endif;
```

2. Adding three extra tabs to the Employee Screen

The `create_panels(pVts:pOutput)` procedure defines the dimensions of the panel area and sets up three distinct tabs:

```
/* ----- */
  Get the field-information
  - and find out if it's an output/protected field
/* ----- */
dcl-proc create_panels;

  dcl-pi *n                pointer;
        pVts              pointer;
        p5250             pointer;
  end-pi;

  dcl-s  pTabPanel         pointer;
  dcl-s  pPanels           pointer;
  dcl-s  pPanelItem        pointer;

  // Create Tab Panel
  pTabPanel = json_newObject();
  json_setStr(pTabPanel : 'xtype': 'tabpanel');
  json_setBool(pTabPanel : 'autoScroll': *off);
  json_setStr(pTabPanel : 'layout': 'fit');
  json_setInt(pTabPanel : 'minHeight': 800);
  json_setInt(pTabPanel : 'minWidth': 1100);
  pPanels = json_newArray();
  json_moveObjectInto(pTabPanel: 'items': pPanels);

  // Tab #1 - Main 5250 + SidePanel/chart
  json_arrayPush(pPanels : create_sidepanel(pVts:p5250));

  // Tab #2 - Internet Search
  pPanelItem = create_panel_internetsearch(pVts);
  json_arrayPush(pPanels : pPanelItem);

  // Tab #3 - A PDF File
  pPanelItem = create_panel_pdf(pVts);
  json_arrayPush(pPanels : pPanelItem);

  return pTabPanel;

end-proc;
```

3. Presenting a Pie Chart using the displayed data on the content tab

This procedure divides the panel area on the first tab into two horizontal sections using the `hbox` layout. The left section contains the 5250 screen with a fixed width of 900px, while the right section, which contains the pie chart, stretches to fill the remaining space. The chart is created using the `create_chart(pVts)` procedure:

```
/* ----- */
  Get the field-information
  - and find out if it's an output/protected field
/* ----- */
dcl-proc create_sidepanel;

  dcl-pi *n                pointer;
        pVts              pointer;
        p5250             pointer;
  end-pi;

  dcl-s  pChart            pointer;
  dcl-s  pItem             pointer;
  dcl-s  pItems            pointer;
  dcl-s  pHbox             pointer;
  dcl-s  pLayout           pointer;
  dcl-s  pPanel            pointer;
  dcl-s  pPanelItems       pointer;

  // Main Panel
  pHbox = json_newObject();
  json_setStr(pHbox : 'title' : 'Employee');
  json_setStr(pHbox : 'xtype' : 'panel');
  pLayout = json_newObject();
  json_setStr(pLayout : 'type' : 'hbox');
  json_setStr(pLayout : 'align' : 'stretch');
  json_moveObjectInto(pHbox : 'layout' : pLayout);

  pPanel = json_newObject();
  json_setStr(pPanel : 'xtype' : 'panel');
  json_setInt(pPanel : 'width' : 900);

  // hbox Left -> 5250
  pPanelItems = json_newArray();
  json_moveObjectInto(pPanel : 'items' : pPanelItems);
  json_arrayPush(pPanelItems : p5250);

  // hbox right -> chart
```

```

pItems = json_newArray();
json_moveObjectInto(pHbox: 'items':pItems);
json_arrayPush(pItems:pPanel);

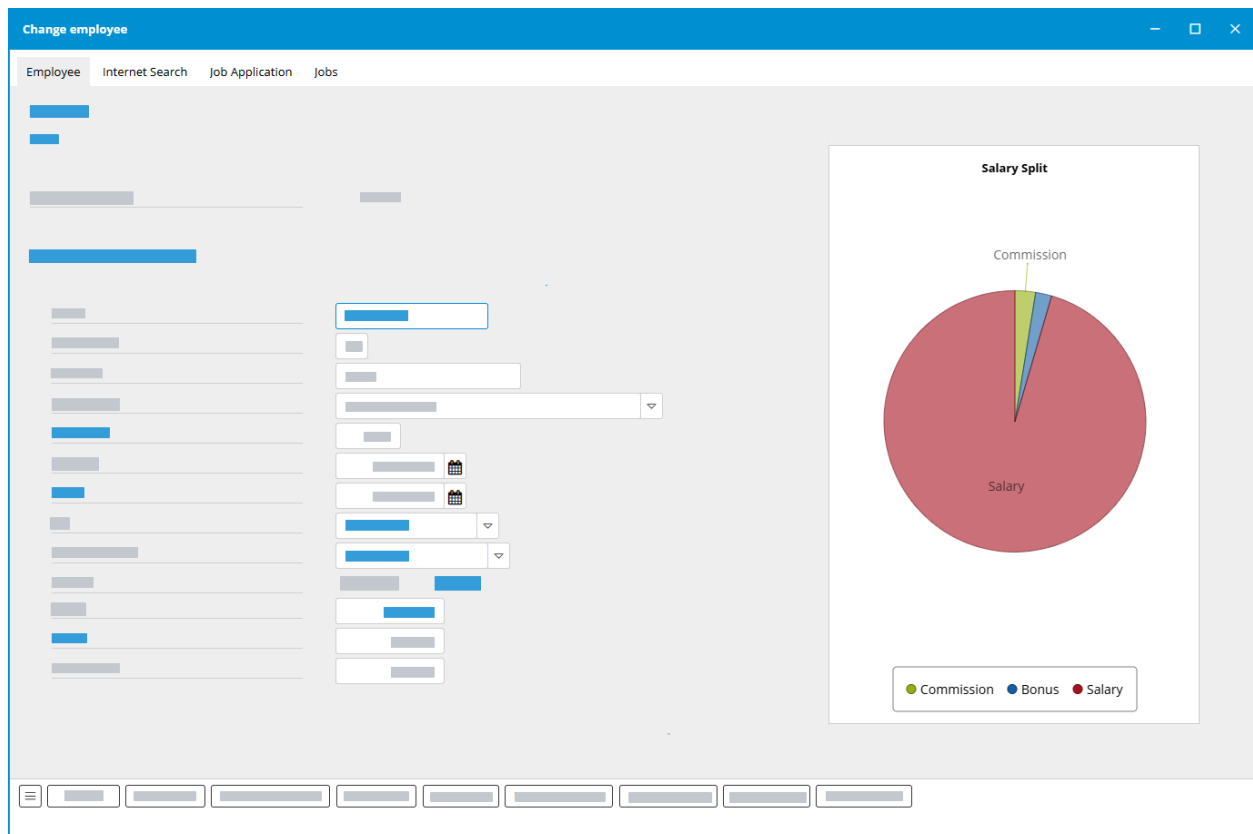
// Create -> Push Chart
pChart = create_chart(pVts);
if pChart <> *null;
    json_arrayPush(pItems:pChart);
endif;

return pHbox;

```

```
end-proc;
```

The pie chart is generated using the salary-related fields (Salary, Bonus, and Commission) from the employee data. The values are retrieved using commands like `vtGetValueFor(pVts: 'Salary')` and converted to integers using `%int()`.



The majority of the statements in this procedure focus on configuring the visual aspects of the chart. In this example, a pie chart is specified using the statement `json_setstr(pSeries: 'type': 'pie')`. Depending on the chart type, the following elements are both common and required:

- **Size:** Defines the width and height of the chart.
- **Style:** Specifies the visual appearance (colors, fonts, line style, etc.).
- **Title:** The main heading that summarizes the chart.
- **Legend:** Describes the meaning of different chart elements.
- **Captions:** Short explanatory text placed below or beside the chart.
- **Series:** Data points plotted on the chart in single or multiple groups.

Refer to [ExtJS Classic Toolkit](#) for more information and different charts.

```
/* ----- */
  Get the field-information
  - and find out if it's an output/protected field
/* ----- */
dcl-proc create_chart;

  dcl-pi *n          pointer;
    pVts            pointer;
  end-pi;

  dcl-s pValue       pointer;
  dcl-s pChart       pointer;
  dcl-s pWrapper     pointer;
  dcl-s pWrapperItems pointer;
  dcl-s pWrapperLayout pointer;
  dcl-s pRight       pointer;
  dcl-s pStore       pointer;
  dcl-s pSeries      pointer;
  dcl-s pTemp        pointer;
  dcl-s pData        pointer;
  dcl-s pCaptions   pointer;
  dcl-s pTitle       pointer;
  dcl-s pStyle       pointer;
  dcl-s salary       int(10);
  dcl-s bonus        int(10);
  dcl-s commission   int(10);

  // Add floating chart component image

  // Title = vtWinTitle(pVts);
  if vtSaaTitleContains(pVts : 'Display employee')
  or vtSaaTitleContains(pVts : 'Change employee');
    pData = json_newArray();
    dcl-s pLegend      pointer;
```

```
pValue = json_newObject();
json_setstr(pValue: 'text': 'Commission');
json_setint(pValue: 'value': %int(vtGetValueFor(pVts: 'Commission')));
json_arrayPush(pData: pValue);
pValue = json_newObject();
json_setstr(pValue: 'text': 'Bonus');
json_setint(pValue: 'value': %int(vtGetValueFor(pVts: 'Bonus')));
json_arrayPush(pData: pValue);
pValue = json_newObject();
json_setint(pValue: 'value': %int(vtGetValueFor(pVts: 'Salary')));
json_setstr(pValue: 'text': 'Salary');
json_arrayPush(pData: pValue);

pChart = json_newObject();
json_setStr(pChart: 'xtype': 'polar');
json_setBool(pChart: 'border': *on);
json_setStr(pChart: 'margin': '60px 60px 60px 0px');
json_setint(pChart: 'flex': 1);
json_setint(pChart: 'innerPadding': 25);

pCaptions = json_newObject();

pStyle = json_newObject();
json_setStr(pStyle: 'fontFamily': 'Open sans');
json_setStr(pStyle: 'fontSize': '13px');
json_setStr(pStyle: 'fontWeight': 'bold');

pTitle = json_newObject();
json_setStr(pTitle: 'text': 'Salary Split');
json_setStr(pTitle: 'align': 'center');
json_setInt(pTitle: 'padding': 15);

json_moveObjectInto(pTitle: 'style': pStyle);
json_moveObjectInto(pCaptions: 'title': pTitle);

// --> panel
pStore = json_newObject();

json_moveObjectInto(pStore: 'data': pData);
json_moveObjectInto(pChart: 'store': pStore);

// --> legend
pLegend = json_newObject();
json_setStr(pLegend: 'docked': 'bottom');
json_setInt(pLegend: 'padding': 15);
json_moveObjectInto(pChart: 'legend': pLegend);

// --> captions
json_moveObjectInto(pChart: 'captions': pCaptions);
```

```
// --> series
pSeries = json_newObject();
json_setStr(pSeries:'type':'pie');
json_setStr(pSeries:'angleField':'value');
json_setBool(pSeries:'highlight':*on);

pTemp = json_newObject();
json_setStr(pTemp:'field':'text');
json_setStr(pTemp:'display':'insideEnd');
json_moveObjectInto(pSeries:'label':pTemp);

pTemp = json_newObject();
json_setDec(pTemp:'opacity':0.6);
json_moveObjectInto(pSeries:'style':pTemp);

// --> panel
json_moveObjectInto(pChart:'series':pSeries);

pWrapper = json_newObject();
json_setStr(pWrapper:'xtype':'panel');
json_setInt(pWrapper:'flex':1);

pWrapperLayout = json_newObject();
json_setStr(pWrapperLayout:'type':'hbox');
json_setStr(pWrapperLayout:'align':'stretch');
json_moveObjectInto(pWrapper:'layout':pWrapperLayout);

pWrapperItems = json_newArray();
json_moveObjectInto(pWrapper:'items':pWrapperItems);
json_arrayPush(pWrapperItems:pChart);

pRight = json_newObject();
json_setStr(pRight:'xtype':'panel');
json_setInt(pRight:'flex':1);

json_arrayPush(pWrapperItems:pRight);

return pChart;
endif;

end-proc;
```

4. Showing the result of a internet search in a tab

In this procedure, the employee's 'Name' and 'Surname' fields are used to form a search string, which is then executed in an embedded iframe in the second tab.

```

/* ----- */
  Get the field-information
  - and find out if it's an output/protected field
/* ----- */
dcl-proc create_panel_internetsearch;

  dcl-pi *n                pointer;
        pVts              pointer;
        scrollable         ind value options(*nopass);
  end-pi;

  dcl-s  w_Scrollable      ind;
  dcl-s  pPanelItem        pointer;
  dcl-s  searchE           varchar(1024);
  dcl-s  empName           varchar(1024);

  w_Scrollable = *off;
  if %parms() >= 2;
    w_Scrollable = scrollable;
  endif;

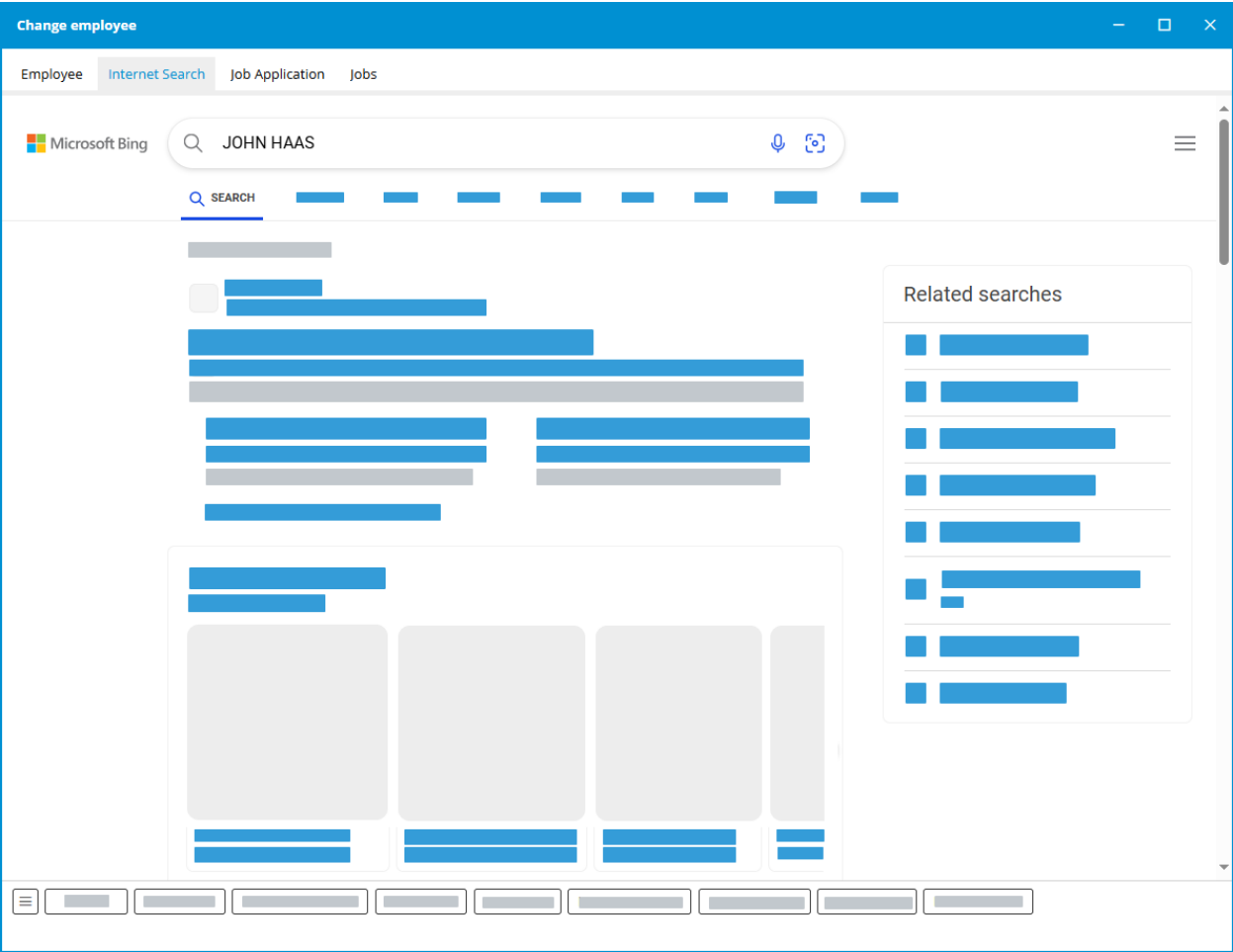
  // Add a Bing-Search for the name
  empName = vtGetValueFor(pVts : 'Name') + ' '
           + vtGetValueFor(pVts : 'Surname');
  searchE = 'http://bing.com/search?q=' + urlEncode(empName);
  searchE = (`
    <iframe src='${searchE}' width=100% height=100% frameborder=0 />
  `);

  // Internet Search
  pPanelItem = json_newObject();
  json_setStr(pPanelItem : 'xtype' : 'panel');
  json_setInt(pPanelItem : 'height' : 1000);
  json_setBool(pPanelItem : 'autoScroll' : w_Scrollable);
  json_setBool(pPanelItem : 'fit' : *on);
  json_setStr(pPanelItem : 'title' : 'Internet Search');
  json_setStr(pPanelItem : 'html' : %trim(searchE));

  return pPanelItem;

end-proc;

```



5. Browse a related PDF document in a tab

This procedure is similar to searching the internet in a tab. The path and file name for the PDF can be dynamically generated based on the employee data.

```

/* ----- */
  Get the field-information
  - and find out if it's an output/protected field
/* ----- */
dcl-proc create_panel_pdf;

  dcl-pi *n                pointer;
        pVts              pointer;
        scrollable         ind value options(*nopass);
  end-pi;

  dcl-s  pPanelItem        pointer;
  dcl-s  pdfFil            varchar(1024);
  dcl-s  w_Scrollable      ind;

  w_Scrollable = *off;
  if %parms() >= 2;
    w_Scrollable = scrollable;
  endif;

  // 3rd tab (Job Application)
  // Add a pdf - Job Application
  pdfFil = 'https://eforms.com/images/2018/03/Simple-Job-Application.pdf';
  pdfFil = (
    <iframe src='${pdfFil}' width=100% height=100% frameborder=0 />
  `);

  pPanelItem = json_newObject();
  json_setStr(pPanelItem : 'xtype': 'panel');
  json_setInt(pPanelItem : 'height': 1000);
  json_setbool(pPanelItem : 'autoScroll': w_Scrollable);
  json_setbool(pPanelItem : 'fit': *on);
  json_setStr(pPanelItem : 'title': 'Job Application');
  json_setStr(pPanelItem : 'html': %trim(pdfFil));

  return pPanelItem;

end-proc;

```

The screenshot shows a web application window titled "Change employee". Inside the window, there are four tabs: "Employee", "Internet Search", "Job Application" (which is active), and "Jobs". Below the tabs is a toolbar with various icons. The main content area displays a form titled "Standard Application for Employment". The form has a header section with a title bar and a toolbar. Below the header, there is a section titled "PERSONAL DATA" which contains several input fields and checkboxes. The form is designed to be flexible and adapt to the available space.

Give the Form Layout a more flexible fit

When introducing additional functions such as chats, browser windows, images, or document viewers within the 5250 screen or as related tabs, a more adaptable layout is often required compared to the default setting of `"formLayout": "absolute"`. To achieve a more flexible layout, you can adjust the configuration by setting `"formLayout": "fit"`. This ensures that the form elements dynamically adjust to the available space, offering a more user-friendly interface.

To implement this, simply add the property to the application call within the menu configuration:

```
{
  "config": "corpflow",
  "func": "call corppgm/employc",
  "initFunc": "",
  "formLayout": "fit"
}
```

Integrating Charts, Internet Searches, and Document Viewing in a Unified Interface

This example illustrates the ability to integrate charts, perform internet searches, and display documents all within the same interface. The entire process is implemented in RPG Free within the Emulation component, running natively on the IBM i platform.

The full source code and examples for these procedures are available in the [/www/iceapp/iceCap2/plugins/em1FlwCorp.rpg1e](#) file and can be tested in the Employee program in the demo environment running in Flow and Flip mode.

Integration

IceCap not only modernizes and optimizes the user interface but also offers significant capabilities for customizing workflows and integrating with other systems and platforms.

This chapter delves into how IceCap can be utilized to enhance IBM i 5250 programs through various integrations. It highlights the following key aspects:

- **IceCap Integration with Web Applications**

This section explains how IceCap facilitates communication between legacy 5250 programs and contemporary web applications, including embedding 5250 programs within web interfaces to enable data exchange between these environments.

- **IceCap as a Web Service for Other Systems**

IceCap can transform traditional 5250 programs into web services, allowing external systems to interact with them. This seamless integration enables external platforms to utilize 5250 program functionalities without needing direct access to the legacy system.

- **IceCap as a Service Layer**

Serving as a middleware, IceCap exposes the business logic and data from 5250 applications as services. This supports a service-oriented architecture (SOA), allowing external applications to access, manipulate, and process data through standardized web services.

These features make IceCap a powerful tool for modernizing IBM i applications by enabling integration with web technologies, enhancing system interoperability, and supporting service-oriented design.

IceCap Application to and from Web Applications

System architecture has evolved from large, monolithic applications confined to single, in-house platforms to a distributed array of microservices operating across multiple platforms within an open architecture, combining both on-premises and cloud environments.

This roadmap outlines how to integrate these differing architectures.

Integration from 5250

Transforming a 5250 program into a web application and migrating it to a browser environment opens up access to various additional services. This transition enables functionalities such as information validation, map integration, and chart generation, which were previously limited by the constraints of the 5250 terminal protocol.

Example:

Consider a scenario where the 5250 interface of an ERP solution has been converted to a web application using IceCap and is running within Portfolio on the IBM i platform. A user needs to evaluate an employee's job title and salary level against a job agency database hosted in the cloud.

The job agency has published an API, available at: <https://jobicy.com/jobs-rss-feed>. Though it does not support direct job title searches, it allows searches based on tags, such as job titles (e.g., "manager"): <https://jobicy.com/api/v2/remote-jobs?count=50&tag=manager>.

In the following example, we demonstrate how to interact with the API and filter search results so that only jobs matching the employee's job title in the database are displayed. The jobs are then presented in a grid within a new tab on the employee's form. This entire process is implemented in RPG Free within the Emulation component, running natively on the IBM i platform.

The examples below can be found in the </www/iceapp/iceCap2/plugins/emlFlwCorp.rpgle>.

1. Recognizing the Employee Screen

The detail screen of the Employee program is identified by the screen title 'Display Employee' or 'Change Employee,' after which a procedure is called to set up additional panels.

```
// Not grid (Subfile)
if (pGrid = *NULL);
  pParams = json_newObject();
  json_setint(pParams : 'startLine' : 1);
  json_setint(pParams : 'endLine' : keyStartLine - 1);
  json_setint(pParams : 'colorTheme' : colorTheme);

  pOutput = icecapCreateFlexLayout(pVts: pParams);
  json_delete(pParams);
```

```

// Work With Employee
// Add Additional 'tabs' for Internet Search and Job application to screen.
if (vtSaaTitleContains(pVts : 'Display employee')
or vtSaaTitleContains(pVts : 'Change employee'));

    // Create additional "Tab Panels"
    pPanel = create_panels(pVts:pOutput);

    return pPanel;

endif;
endif;

```

2. Adding Extra Panels to the Employee Screen

This procedure defines the panel area and sets up four tabs. Let's dive into the fourth tab, which shows job listings.

```

/* ----- */
Get the field-information
- and find out if it's an output/protected field
/* ----- */
dcl-proc create_panels;

    dcl-pi *n                pointer;
           pVts             pointer;
           p5250            pointer;
    end-pi;

    dcl-s  pTabPanel         pointer;
    dcl-s  pPanels           pointer;
    dcl-s  pPanelItem        pointer;

    // Create Tab Panel
    pTabPanel = json_newObject();
    json_setStr(pTabPanel : 'xtype': 'tabpanel');
    json_setbool(pTabPanel : 'autoScroll': *off);
    json_setStr(pTabPanel : 'layout': 'fit');
    json_setInt(pTabPanel : 'minHeight': 800);
    json_setInt(pTabPanel : 'minWidth': 1100);
    pPanels = json_newArray();
    json_moveObjectInto(pTabPanel: 'items': pPanels);

    // Tab #1 - Main 5250 + SidePanel/chart
    json_arrayPush(pPanels : create_sidepanel(pVts:p5250));

```

```

// Tab #2 - Internet Search
pPanelItem = create_panel_internetsearch(pVts);
json_arrayPush(pPanels :pPanelItem);

// Tab #3 - A PDF File
pPanelItem = create_panel_pdf(pVts);
json_arrayPush(pPanels :pPanelItem);

// Tab #4 - Jobs from API
pPanelItem = create_panel_jobs(pVts);
json_arrayPush(pPanels :pPanelItem);

return pTabPanel;

end-proc;

```

3. Setting Up the Tab and Panel for Jobs from the Agency

This procedure defines the columns and fields, creates the data store, and builds the grid layout including error handling.

```

/* ----- */
  Get the field-information
  - and find out if it's an output/protected field
/* ----- */
dcl-proc create_panel_jobs;

  dcl-pi *n                pointer;
    pVts                  pointer;
  end-pi;

  dcl-s  pPanel            pointer;
  dcl-s  items             pointer;
  dcl-s  pGrid             pointer;
  dcl-s  pStore            pointer;
  dcl-s  pStoreFields      pointer;
  dcl-s  pStoreField       pointer;
  dcl-s  pColumns          pointer;
  dcl-s  pColumn           pointer;
  dcl-s  pData             pointer;
  dcl-s  pResult           pointer;
  dcl-s  job               varchar(8);
  dcl-s  json_text         varchar(8192);

  pPanel = json_newObject();
  json_setStr(pPanel: 'xtype': 'panel');

```

```
json_setStr(pPanel: 'layout': 'fit');
json_setInt(pPanel: 'flex': 1);
json_setStr(pPanel: 'title': 'Jobs');

// Add a Bing-Search for the name
job = vtGetValueFor(pVts : 'Job');
pResult = getGridData(job);

pData = json_locate(pResult: 'data');

if pData <> *null;

    // CREATE COLUMNS
    pColumns = json_newArray();

    pColumn = json_newObject();
    json_setStr(pColumn: 'text': 'Title');
    json_setStr(pColumn: 'dataIndex': 'jobTitle');
    json_setInt(pColumn: 'flex': 3);

    json_arrayPush(pColumns: pColumn);

    pColumn = json_newObject();
    json_setStr(pColumn: 'text': 'Min Salary');
    json_setStr(pColumn: 'xtype': 'numbercolumn');
    json_setStr(pColumn: 'format': '0,000');
    json_setStr(pColumn: 'dataIndex': 'annualSalaryMin');
    json_setStr(pColumn: 'align': 'right');
    json_setInt(pColumn: 'flex': 1);

    json_arrayPush(pColumns: pColumn);

    pColumn = json_newObject();
    json_setStr(pColumn: 'xtype': 'numbercolumn');
    json_setStr(pColumn: 'format': '0,000');
    json_setStr(pColumn: 'text': 'Max Salary');
    json_setStr(pColumn: 'dataIndex': 'annualSalaryMax');
    json_setStr(pColumn: 'align': 'right');
    json_setInt(pColumn: 'flex': 1);

    json_arrayPush(pColumns: pColumn);

    pColumn = json_newObject();

    json_setStr(pColumn: 'text': 'Currency');
    json_setStr(pColumn: 'dataIndex': 'salaryCurrency');
    json_setStr(pColumn: 'align': 'center');
    json_setInt(pColumn: 'flex': 1);

    json_arrayPush(pColumns: pColumn);
```

```
pColumn = json_newObject();
json_setStr(pColumn: 'xtype': 'datecolumn');
json_setStr(pColumn: 'format': 'Y-m-d');
json_setStr(pColumn: 'text': 'Published');
json_setStr(pColumn: 'dataIndex': 'pubDate');
json_setStr(pColumn: 'align': 'center');
json_setInt(pColumn: 'flex': 1);

json_arrayPush(pColumns: pColumn);

pColumn = json_newObject();
json_setStr(pColumn: 'xtype': 'linkcolumn');
json_setStr(pColumn: 'text': 'Link');
json_setStr(pColumn: 'dataIndex': 'url');
json_setStr(pColumn: 'align': 'center');
json_setInt(pColumn: 'width': 100);

json_arrayPush(pColumns: pColumn);

// Create fields
pStoreFields = json_newArray();

pStoreField = json_newObject();
json_setStr(pStoreField: 'type': 'auto');
json_setStr(pStoreField: 'name': 'jobTitle');

json_arrayPush(pStoreFields: pStoreField);

pStoreField = json_newObject();
json_setStr(pStoreField: 'type': 'auto');
json_setStr(pStoreField: 'name': 'url');

json_arrayPush(pStoreFields: pStoreField);

// CREATE STORE
pStore = json_newObject();

json_moveObjectInto(pStore: 'fields': pStoreFields);
json_moveObjectInto(pStore: 'data': pData);

// CREATE GRID
pGrid = json_newObject();

json_setStr(pGrid: 'xtype': 'grid');
json_setStr(pGrid:
    'emptyText':
        `No jobs found for: <strong>${job}</strong>`
);
```

```

    json_moveObjectInto(pGrid: 'columns': pColumns);
    json_moveObjectInto(pGrid: 'store': pStore);

    items = json_newArray();

    json_arrayPush(items: pGrid);

    json_moveObjectInto(pPanel: 'items': items);

else;
    json_setInt(pPanel : 'padding':10);
    json_setStr(pPanel : 'html': json_getStr(pResult: 'error'));
endif;

// Panel Jobs

return pPanel;

end-proc;

```

4. Retrieving and Filtering Jobs from the API

This procedure retrieves up to 50 job entries from the API and filters them to only show jobs where the job title is part of the employee's job title.

```

/* -----
   Get Job Data from the API
   ----- */
dcl-proc getGridData;

    dcl-pi *n                pointer;
           job              varchar(8) value;
    end-pi;

    dcl-s  pOutput          pointer;
    dcl-s  pResult          pointer;
    dcl-s  pJobs            pointer;
    dcl-s  pRows            pointer;
    dcl-s  pRequest         pointer;
    dcl-s  pResponse        pointer;
    dcl-s  url              varchar(256);
    dcl-s  server           varchar(256);
    dcl-s  error            ind;
    dcl-s  print            ind;
    dcl-s  statusCode       int(5);
    dcl-s  cnt              int(10);

```

```
dc1-s    c                int(10);
dc1-s    temp             varchar(128);
dc1-s    json_text        varchar(8192);
dc1-s    title            varchar(256);
dc1-ds   itJobs           likeds(json_iterator);

pRequest = sesGetILOB('request');
pResponse = sesGetILOB('response');

ilob_clear(pRequest);
ilob_clear(pResponse);

print = *on;

job = %trim(job);

server = 'https://jobicy.com/api/v2/remote-jobs?count=50';

url = (
{
    request : {
        url: "${server}&tag=${job}",
        method: "GET",
        timeout: 30
    }
}
);

error = httpRequest2(
    url
    :*NULL
    :*OMIT
    :*OMIT
    :pRequest
    :*OMIT
    :*OMIT
    :pResponse
);
if (error);
    pOutput = json_newObject();
    json_setBool(pOutput : 'success' : *OFF);
    json_setstr(
        pOutput
        : 'error'
        : 'Error ${server} did not answer correct'
    );
else;
    pOutput = json_newObject();
    json_setBool(pOutput : 'success' : *ON);
    pResult = json_ParseILOB(pResponse: 'syntax=loose':1208:0);
```

```
pJobs = json_locate(pResult: 'jobs');

if pJobs <> *null;

    statusCode = json_getInt(pJobs: 'statusCode': 0);

    if statusCode > 400;
        json_setStr(pOutput: 'error': json_getStr(pJobs: 'message'));
    else;

        pRows = json_newArray();

        itJobs = json_setIterator(pJobs: '');

        dow json_forEach(itJobs);

            title = json_getStr(itJobs.this: 'jobTitle': '');

            job = %trim(LowerCase(job));
            title = LowerCase(title);

            cnt = words(title: ' ');

            for c = 1 to cnt;

                temp = %trim(word(title: c: ' '));

                if temp = job;
                    json_arrayPush(pRows: itJobs.this);
                    leave;
                endif;

            endfor;

        enddo;

        json_moveObjectInto(pOutput : 'data' : pRows);
    endif;

    if (print);
        json_text = JSON_ASJSONTEXT16M(pRows);
    endif;
else;
    json_setStr(pOutput: 'error': json_getStr(pResult: 'error'));
endif;

endif;

ilob_delete(pRequest);
ilob_delete(pResponse);
```

```
return pOutput;  
  
end-proc;
```

Remember to compile the program after making changes:
`CRTICEPGM STMF('/iceapp/iceCap2/plugins/emlFlwCorp.rpgle') SVRID(<Portfolio-Server>)`
`LIBL(*SERVER)`

The new tab

The new "Jobs" tab allows the user, with a single click, to evaluate the employee's salary level and view detailed job advertisements directly within the browser.

Change employee

Employee

Internet Search

Job Application

Jobs

Title	Min Salary	Max Salary	Currency	Published	Link

Integration to 5250

Accessing a specific 5250 screen from another web application or platform involves several steps that are challenging for many conventional tools, such as:

Challenge	Description
Authentication	Verifying user credentials to ensure secure access.
Menu Navigation	Navigating complex menu structures to reach the desired 5250 screen.
Program Launching	Initiating the 5250 application accurately.
Screen Navigation	Progressing through intermediate 5250 screens efficiently.
Screen Display	Ensuring a compatible display format for web use.

These requirements form a typical 5250 user workflow, revealing the difficulties standard tools face with such tasks.

IceCap simplifies this entire process. Its Emulation and Navigation components replicate the user journey by automating steps and bypassing intermediary screens to reach the target screen directly. IceCap then displays the final screen in a web-ready format, making it accessible from most platforms and standard web environments.

Use Case Example

Imagine a scenario where a user needs to view or edit employee information within an ERP system on the IBM i platform but does not have direct access. Working on a different platform, the user would benefit from accessing this information with a single click.

This example demonstrates how to address authentication, initiate a 5250 program, and navigate to a specific employee display/edit screen based on provided parameters.

1. Authentication and Program Launching

In this scenario, we employ the auto-login feature with a specific presentation layout to streamline access. This setup ensures only IceCap is displayed, while Portfolio operates in the background without appearing onscreen. Before proceeding, please review the "Auto Login and Layout

Customization" section in the "Getting Started" chapter to understand the capabilities of these features.

Using this approach, we bypass the login screen by embedding the username and password directly in the URL. Additionally, we suppress all other Portfolio components by selecting a layout (e.g., layout 30). To integrate seamlessly with the user's existing graphical interface on the other platform, we use the "corpflow" presentation mode:

```
http://myIBMi:7700/?layout=31&username=demo&password=DEMO#IceCap.view.Main-{"func":"call
corppgm/employc","options":{"config":"corpflow"},"parms":{"employid":"000050","option":"2"}}
```

Before using this method, be aware of the security implications of sending credentials through a URL. Additionally, the use of this feature must be confirmed by setting 'autoLigon: true' in the configuration manifest file located at `/www/iceapp/<portfolio>/manifest.js`. For more details about layouts, refer to the "Linking to Applications in Portfolio" chapter.

2. Screen Navigation

The following example demonstrates navigating through 5250 screens by identifying text at specific positions. Here, the program detects the presence of 'Corp. Data' in line 1, position 2, with a length of 10 characters, and 'EmpNo.' in line 9, position 8, with a length of 6 characters.

Corp. Data

Work with employees

MHO

Start with Employee Number

Type options, press Enter.

1=Create 2=Change 3=Copy 4=Delete 5=Display 8=Activities

Opt	EmpNo.	Name	Surname	Dept.	Phone	Job	Level	Birthday
2	000123							
—	000010	JOHN	HAAS	A00	3425	ANALYST	13	12-01-1965
—	000020	MICHAEL	THOMPSON	C01	3476	CTO	14	02-02-1948

This code snippet is part of the navigation program `navCapCorp.rpgle`, designated as the Navigation program for the `corpFlow` presentation mode in the `icecap_config.xml` configuration file. The source code is available in the `/iceapp/icecap2/plugins/` folder on the IFS.

```
Select;
  // Employees
```

```
when vtGetScreen(pVts:1:2:10) = 'Corp. Data'
and vtGetScreen(pVts:9:8:6) = 'EmpNo.';

// Get employ id from parameters
wValue = json_getstr(vts.pParms : 'employid');

// If we have a value perform action
if wValue > '';
    wOption = json_getstr(vts.pParms : 'option': '5');
    vtSetFieldNo(pVts:2:wOption);
    vtSetFieldNo(pVts:3:wValue);
    vtPressKey(pVts: vtENTER);
endif;

EndS1;
```

The navigation program monitors the data stream and is prepared to intervene when the employee subfile screen appears. This intervention occurs only if parameters are attached, including one named **employid**.

When detected, the Navigation program automatically enters the option number (e.g., 2 for "Change") and the specified Employee ID (e.g., 000050) and presses **[ENTER]** to access the Display or Change Employee screen.

3. Screen Display

Upon entering the Employee Detail screen, the Navigation Program activates the Virtual Terminal display by setting **vts.showScreen = *on**, making the screen visible to the user.

```
vts.showScreen = *on;
```

The user can then update employee information as needed. By pressing **[Enter]**, changes are saved, or by pressing **[F3]**, the user can exit without saving..

Employee

Internet Search

Job Application

Jobs

Corp. Data

MHO

Employee Number

000050

Type information, press Enter.

Name

JOHN

Midle Name

B

Surname

GEYER

Department

E01 - SUPPORT SERVICES

Phone No.

6789

Birthday

01-09-1925

Hired

17-08-1949

Job

MANAGER

IceCap as a Web Service for Other System

In the previous example, we focused on displaying data directly on the screen, a model called "integration on the glass," where information from multiple sources is visually presented without being stored or merged. Now, we will demonstrate how to move from this visual integration approach to creating a web service that operates efficiently in the background, handling transactions without an on-screen display.

Example

Suppose a program on another platform needs to update an employee's "Job Title" or "Department." In this case, the 5250 screen should not display, and any errors should return a response indicating success or failure, such as:

```
{"success":true}
```

```
{"success":false,"message":"Department not found"}
```

Implementing the Web Service

We will create a web service named `CorpApi1.rpgle` with three parameters: `action` (e.g., update), `job` (e.g., MANAGER), and `department` (e.g., D11). This service will perform the updates in the background, providing a streamlined and automated transaction solution.

The service can be tested by accessing the following URL in a browser:

```
http://myIBMi:7700/icecap/corpApi1.rpgle?  
action=update&employeeNumber=000010&job=MANAGER&department=A00
```

Important: The profile in this service is set to log in as the user DEMO with the password DEMO. Ensure this user is created on the system, or update the service program with the appropriate user profile and password before testing

This program flow follows a similar structure to previous examples but utilizes different programming and operational methods:

1. Initialization and message handling

The following RPG code initiates the service, processes incoming messages, and handles input data:

```

**free
ctl-opt copyright('System & Method (C), 2024');
ctl-opt decEdit('0,') datEdit(*YMD.) main(main);
ctl-opt text('IceCap Intergration: Example 1');
ctl-opt bndDir('ICECAPTERM');

/include qrppleref,icecapTerm
/include qasphdr,jsonparser

/* -----
   Main line
   ----- */
dcl-proc main;

    dcl-pi *n;
    end-pi;

    dcl-s pInput          pointer;
    dcl-s pOutput         pointer;
    dcl-s pVts            pointer;
    dcl-s key              varchar(256);
    dcl-s value            varchar(256);

    setContentType('application/json; charset=utf-8');
    setEncodingType('*JSON');

    pVts = login();
    if (pVts = *NULL);
        return;
    endif;

    navigateToCorpEmployee(pVts);
    if (not vtSaaTitleContains(pVts:'Work with employees'));
        sendMessage('Unable to find screen: Work with employees');
        return;
    endif;

    // Get input query string
    pInput = json_newObject();
    getQryStrList(key :value :'*FIRST');
    dow (key > '');
        json_setValue(pInput :key :value);
        getQryStrList(key :value :'*NEXT');
    enddo;

    select;
    when json_getstr(pInput :'action') = 'update';

```

```

        changeEmployeeDepartment(pVts :pInput);
    other;
        vtClose(pVts);
        sendMessage('Invalid action');
        return;
    ends1;

    vtClose(pVts);

    pOutput = json_newObject();
    json_setbool(pOutput : 'success' : *ON);
    responseWriteJson(pOutput);
    json_delete(pOutput);

    return;

end-proc;

```

If the service program or the 5250 screen encounters an error, the `sendMessage` procedure will pass a status of `"success" : false` along with an error message in the web service response:

```

/* -----
   Send message
   ----- */
dcl-proc sendMessage;

    dcl-pi *n;
        message                varchar(256) value;
    end-pi;

    dcl-s pOutput                pointer;

    pOutput = json_newObject();
    json_setbool(pOutput : 'success' : *OFF);
    json_setstr(pOutput : 'message' : message);
    responseWriteJson(pOutput);
    json_delete(pOutput);

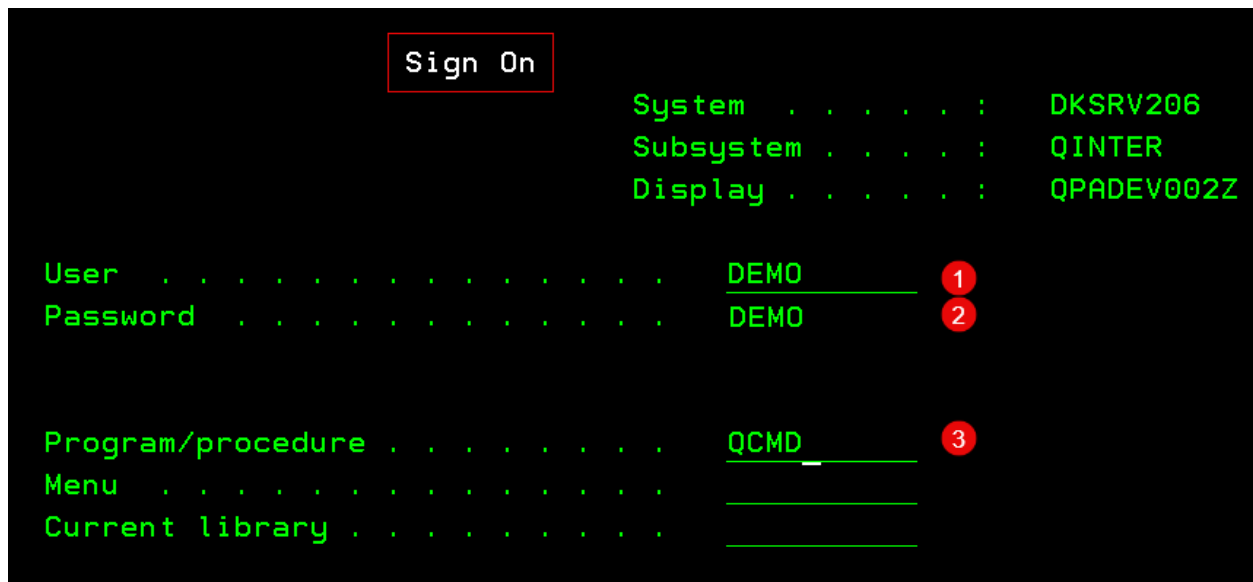
    return;

end-proc;

```

2. Authentication

The `login` procedure identifies the login screen by its title, "Sign On," fills in the first three fields (user profile, password, and program), and presses [ENTER] to proceed



If certain intermediate screens appear, the procedure bypasses them by pressing [ENTER] again:

```
vtSaaTitleContains(pVts:'Sign-on Information')
vtSaaTitleContains(pVts:'Attempt to Recover Interactive Job')
vtSaaTitleContains(pVts:'Display Program Messages')
```

When the "Command Entry" screen is detected (`vtSaaTitleContains(pVts:'Command Entry')`), the procedure exits to the main program. Any error messages are passed to the message handler before completing the authentication.

```
/* -----
   Start Virtual terminal, login and make sure to end on main screen
   ----- */
dcl-proc login;

    dcl-pi *n                pointer;
    end-pi;

    dcl-s pVts                pointer;
    dcl-ds vts                likeds(vtsDS) based(pVts);
    dcl-ds vtLogon            likeds(vtLogonDS);
    dcl-s screenFound        ind;

    // Open IceCap Virtual Terminal Session
```

```

vtLogon = *LOVAL;
vtLogon.keepInput = *ON;
pVts = vtOpen(vtLogon);

screenFound = *OFF;
do (screenFound);
    select;
    // Logon if the signon is displayed
    when (vtSaaTitleContains(pVts:'Sign On'));
        vtSetFieldNo(pVts: 1 : 'DEMO');    // First field is the userprofile
        vtSetFieldNo(pVts: 2 : 'DEMO');    // Second field is the password
        vtSetFieldNo(pVts: 3 : 'QCMD');    // Program to run - QCMD
        vtPressKey(pVts: vtENTER);
        // Check message
        if (vts.errMsg <> '');

        sendMessage(vts.errMsg);
        vtClose(pVts);
        return *NULL;
    endif;
    if (vtWinGetLine(pVts :24) <> ''
    and %subst(vtWinGetLine(pVts :24) :1 :3) = 'CPF');
        sendMessage(vtWinGetLine(pVts :24));
        vtClose(pVts);
        return *NULL;
    endif;
    when (vtSaaTitleContains(pVts:'Sign-on Information')
    or   vtSaaTitleContains(pVts:'LOG PÅ'));
        if ((vtWinGetLine(pVts :3) <> ''
        and %subst(vtWinGetLine(pVts :3) :2 :10) <> 'last login')
        or (vtWinGetLine(pVts :3) <> ''
        and %subst(vtWinGetLine(pVts :3) :2 :21) <> 'Logget på sidste gang'));
            sendMessage(vtWinGetLine(pVts :3));
            vtClose(pVts);
            return *NULL;
        else;
            vtPressKey(pVts: vtENTER);
        endif;
    when (vtSaaTitleContains(pVts:'Attempt to Recover Interactive Job'));
        vtSetFieldNo(pVts: 1 : '90');
        vtPressKey(pVts: vtENTER);
    when (vtSaaTitleContains(pVts:'Display Program Messages')
    or   vtSaaTitleContains(pVts:'VIS PROGRAMMEDDELELSER'));
        vtPressKey(pVts: vtENTER);
    when (vtSaaTitleContains(pVts:'Command Entry'));
        screenFound = *ON;
    endl;
enddo;

return pVts;

```

```
end-proc;
```

3. Navigating the menu system and Launching the 5250 program

To initiate navigation from the "Command Entry" screen, the program issues the following IBM i commands:

```
when (vtSaaTitleContains(pVts:'Command Entry'));
    vtSetFieldNo(pVts: 1 : 'go corppgm/corpmnu');
    vtPressKey(pVts: vtENTER);
```

This example demonstrates how easily IceCap VT commands integrate into an RPG program to navigate menus and launch applications.

```
/* -----
   Navigate to Corp. employee screen
   ----- */
dcl-proc navigateToCorpEmployee;

    dcl-pi *n;
        pVts                pointer;
    end-pi;

    dcl-ds vtLogon            likeds(vtLogonDS);
    dcl-s  screenFound        ind;

    screenFound = *OFF;
    dou (screenFound);
        select;
            when (vtSaaTitleContains(pVts:'Command Entry'));
                vtSetFieldNo(pVts: 1 : 'go corppgm/corpmnu');
                vtPressKey(pVts: vtENTER);
            when (vtSaaTitleContains(pVts:'Corp. Data Main Menu'));
                vtSetFieldNo(pVts: 1 : '1');
                vtPressKey(pVts: vtENTER);
            when (vtSaaTitleContains(pVts:'Work with employees'));
                screenFound = *ON;
            ends1;
        enddo;
end-proc;
```

4. Navigating through 5250 screens

In the code example below, the program navigates to the screen titled "Work with employees," enters "2" in the second input field labeled "Opt," and "000050" in the third input field labeled "EmpNo," then presses [ENTER] to proceed:

```

Corp. Data                                     Work with employees
MHO

Start with . . . . . 1 Employee Number

Type options, press Enter.
  1=Create   2=Change   3=Copy   4=Delete   5=Display   A=Activities

Opt  EmpNo.  Name      Surname      Dept.  Phone  Job      Level  Bir
  2  2  000050  3
  —  —  000010  ELON        HAAS        A00  3425  IT        12  12-01
  —  —  000020  MICHAEL     THOMPSON    C01  3476  CTO       14  02-02

```

On the subsequent screen, titled "Change Employee," data is entered using two methods:

- **Field Sequence Method**

The Department ID is entered in a fixed input field position (field number 4).

- **Label-Based Method**

The Job Title field is identified by its label ("Job").

While the field sequence method is faster and involves fewer lines of code, it should only be used if field order remains unchanged. The label-based method is more flexible and can adapt even if field positions change, as long as the label is consistent.

Corp. Data MHO	Change employee	
Employee Number : 000050		
Type information, press Enter.		
Name	JOHN	1
Midle Name	B	2
Surname	GEYER	3
Department	E01	4 SUPPORT SERVICES
Phone No.	6789	
Birthday	01-09-1925	(DMYY)
Hired	17-08-1949	(DMYY)
Job	MANAGER	(JOB)
Education Level	13	(12-20)

```

/* -----
Change employee
----- */
dcl-proc changeEmployeeDepartment;

dcl-pi *n;
    pVts                pointer value;
    pInput              pointer value;
end-pi;

dcl-ds vts              likeds(vtsDS) based(pVts);
dcl-ds location         likeds(vtLocationDS);

// Go into change employee
vtSetFieldNo(pVts :2 :'2'); // Field 2 is first option field
vtSetFieldNo(pVts :3 :json_getstr(pInput :'employeeNumber'));
vtPressFormKey(pVts: 'ENTER');
if (not vtSaaTitleContains(pVts:'Change employee'));
    sendMessage('Unable to find screen: Change employees');
    return;
endif;

// Change employee name by finding the label
if (json_has(pInput :'job'));
    location = vtLocate(
        pVts
        :VT_INPUT
        :vtLinCol(1:1)
        :VT_EAST
        :'Job . . . . .')

```

```
    );
    if (location.lin > 0);
        vtSetField(
            pVts
            :location.lin
            :location.col
            :json_getstr(pInput : 'job')
        );
    endif;
endif;

// Change department by known field no.
if (json_has(pInput : 'department'));
    vtSetFieldNo(pVts :4 :json_getstr(pInput : 'department'));
endif;

vtPressFormKey(pVts: 'ENTER');

// Still at change employee, check for error message
if (vtSaaTitleContains(pVts: 'Change employee'));
    if (vts.errMsg <> '');
        sendMessage(vts.errMsg);
    else;
        if (vtWinGetLine(pVts :24) <> '');
            sendMessage(vtWinGetLine(pVts :24));
        endif;
    endif;
endif;

// Exit work with employess
vtPressFormKey(pVts: 'F3');

return;

end-proc;
```

IceCap as a Service Layer

Microservices architecture, characterized by decomposing applications into smaller, self-contained services, is distinct from the monolithic structure typically found in 5250 environments. Converting a 5250 program into a set of smaller web services presents challenges. However, IceCap simplifies this process by removing the need to modify the 5250 program or its source code; the original program continues to function as it always has.

For the Employee program, transforming it into a web service involves creating four core operations as RESTful services:

HTTP Method	Procedure	Description
GET	GETEMPLOYEE	Read Employee Data
POST	CREATEEMPLOYEE	Create Employee Data
PUT	UPDATEEMPLOYEE	Update Employee Data
DELETE	DELETEEMPLOYEE	Delete Employee Data

The goal is to establish a service layer that fully exposes the Employee program's functionality as RESTful web services, replicating the capabilities of the 5250 program without additional business logic.

Router Program for RESTful Services

To manage these services, a router program (`corpRouter.rpg1e`) will be created. This router directs HTTP requests to the appropriate service endpoint based on the request's path and method (e.g., GET, POST), effectively organizing and separating routing from business logic. Routers are essential in RESTful architectures as they centralize error handling, support scalability, and simplify middleware integration for tasks like authentication. Over time, `corpRouter.rpg1e` can be extended to cover all programs in the demo environment, CorpData, beginning with the Employee program routes in this example.

1. Initiation and Main Process

Sets up the main program environment, including initialization tasks and launching the primary process flow. This step prepares the program to handle incoming requests efficiently.

```

**free
// Require routing entry in webconfig.xml
// Ex. <map pattern="corporate/*" pgm="icecapRout" lib="*LIBL"/>

ctl-opt copyright('System & Method (C), 2024');
ctl-opt decEdit('0,') datEdit(*YMD.) main(main);
ctl-opt text('Icecap Router: Example 2');

/include noxdb
/include qasphdr,iceutility
/* -----
   Main line:
   ----- */
dcl-proc main;

    dcl-s pInput          pointer;
    dcl-s method          varchar(32);

    setContentType('application/json; charset=utf-8');
    setEncodingType('*JSON');

    method = strUpper(getServerVar('REQUEST_METHOD'));
    // Ignore options request
    if (method = 'OPTIONS');
        setStatus('200');
        return;
    endif;

    pInput = getInputParameter();

    processRequest(pInput);

    json_delete(pInput);

    return;

end-proc;

```

2. Process Action

Handles the specific action requested by the user, directing the flow to the appropriate service or operation based on the input action type.

```

/* -----
   Process action
   ----- */

```

```

dcl-proc processRequest;

    dcl-pi *n;
        pInput                pointer value;
    end-pi;

    dcl-s pOutput                pointer;

    pOutput = callService(pInput);

    if (pOutput <> *NULL);
        responseWriteJson(pOutput);
        json_close(pOutput);
    endif;

end-proc;

```

3. Call Service Program

Invokes the service program responsible for executing core operations, ensuring the requested tasks are performed as specified.

```

/* -----
Call service program
Request pattern: /system/module/{id}
----- */
dcl-proc callService;

    dcl-pi *n pointer;
        pInput                pointer value;
    end-pi;

    dcl-pr ActionProc          pointer extproc(pProc);
        payload                pointer value;
    end-pr;

    dcl-s pOutput                pointer;
    dcl-s pResponse              pointer;
    dcl-s pProc                  pointer (*PROC) static;
    dcl-s action                 varchar(8192);
    dcl-s system                 varchar(256);
    dcl-s pgmName                 char(10);
    dcl-s procName                varchar(128);
    dcl-s errorText               char(128);
    dcl-s errorPgm                char(64);
    dcl-s errorList               char(4096);

```

```
action = strUpper(getServerVar('REQUEST_FULL_PATH'));
system = word(action :1 :'/');
select;
when system = 'CORPDATA';
    pOutput = setupCorporate(pInput :action);
other;
    pResponse = formatErrorMessage(
        'Invalid request pattern'
    );
    return pResponse;
endsl;

if (json_getstr(pOutput : 'success') = 'false');
    return pOutput;
endif;

pgmName = json_getstr(pOutput : 'programName');
procName = json_getstr(pOutput : 'procedureName');

pProc = loadServiceProgramProc('*LIBL':pgmName :procName);
if (pProc = *NULL);
    pResponse = formatErrorMessage(
        'Invalid action:' + action + ' or service not found'
    );
else;
    monitor;
        pResponse = ActionProc(pInput);
    on-error;
        setStatus ('500');
        soap_Fault(errorText:errorPgm:errorList);
        pResponse = formatErrorMessage(
            'Error in service ' + action + ', ' + errorText
        );
    endmon;

    if (pResponse = *NULL);
        setStatus('404');
        return pResponse;
    endif;

    if (json_getstr(pResponse : 'success') = 'false' and json_has(pResponse : 'message'));
        if (json_has(pResponse : 'statusCode'));
            setStatus(
                json_getstr(pResponse : 'statusCode')
                + ' '
                + json_getstr(pResponse : 'message')
            );
        else;
```

```

        setStatus('406 ' + json_getstr(pResponse : 'message'));
    endif;
endif;
endif;

return pResponse;

end-proc;

```

4. Setup Service Program and Procedure for “Corporate” System

Configures the service program and procedure for the “Corporate” system, establishing necessary settings and routines to integrate with corporate data and procedures.

```

/* -----
Setup service program and procedure for Corporate System
----- */
dcl-proc setupCorporate;

    dcl-pi *n                pointer;
        pInput              pointer;
        action              varchar(8192);
    end-pi;

    dcl-s pOutput            pointer;
    dcl-s module            varchar(256);
    dcl-s id                varchar(256);

    module = word(action :2 :'/');

    select;
    when module = 'EMPLOYEE';
        pOutput = json_newObject();
        json_setstr(pOutput : 'programName' : 'CORPAPI2'); // IceCapIntegration example #2
        select;
        when getServerVar('REQUEST_METHOD') = 'GET';
            json_setstr(pOutput : 'procedureName' : 'GETEMPLOYEE');
            json_setstr(pInput : 'EMPLOYEEENUMBER' : word(action :3 :'/'));
        when getServerVar('REQUEST_METHOD') = 'POST';
            json_setstr(pOutput : 'procedureName' : 'CREATEEMPLOYEE');
        when getServerVar('REQUEST_METHOD') = 'PUT';
            json_setstr(pOutput : 'procedureName' : 'UPDATEEMPLOYEE');
            json_setstr(pInput : 'EMPLOYEEENUMBER' : word(action :3 :'/'));
        when getServerVar('REQUEST_METHOD') = 'DELETE';
            json_setstr(pOutput : 'procedureName' : 'DELETEEMPLOYEE');
            json_setstr(pInput : 'EMPLOYEEENUMBER' : word(action :3 :'/'));

```

```

        other;
        json_delete(pOutput);
        pOutput = formatErrorMessage('Invalid method');
    ends1;
other;
    pOutput = formatErrorMessage('Invalid module');
ends1;

return pOutput;

end-proc;

```

5. Get Data from Request

Extracts data from the incoming request, ensuring the correct information is captured and ready for processing in the relevant services.

```

/* -----
get data from request
----- */
dcl-proc getInputParameter;

    dcl-pi *n                pointer;
    end-pi;

    dcl-s pOutput            pointer;
    dcl-s pMessage           pointer;
    dcl-s key                varchar(256);
    dcl-s value              varchar(256);
    dcl-s pOptions           pointer static;

    if (pOptions = *NULL);
        pOptions = json_newObject();
        json_setBool(pOptions : 'upperCaseColname' : *OFF);
        json_setBool(pOptions : 'autoParseContent' : *ON);
        json_setBool(pOptions : 'sqlNaming'         : *OFF);
    endif;

    json_sqlSetOptions(pOptions);

    if reqStr('payload') > '';
        pOutput = json_parseString(reqStr('payload'));
    elseif getServerVar('REQUEST_METHOD') = 'POST';
        pOutput = json_parseRequest();
    elseif getServerVar('REQUEST_METHOD') = 'PUT';
        pOutput = json_parseRequest();
    endif;
end-proc;

```

```

elseif getServerVar('REQUEST_METHOD') = 'PATCH';
    pOutput = json_parseRequest();
elseif getServerVar('REQUEST_METHOD') = 'DELETE';
    pOutput = json_parseRequest();
elseif getServerVar('REQUEST_METHOD') = 'GET';
    pOutput = json_newObject();
    getQryStrList(key :value :'*FIRST');
    dow (key > '');
        json_setValue(pOutput :key :value);
        getQryStrList(key :value :'*NEXT');
    enddo;
else;
    pOutput = *NULL;
endif;

if (pOutput <> *NULL and json_error(pOutput));
    pMessage = formatErrorMessage('Please POST payload in JSON');
    responseWriteJson(pMessage);
    json_delete(pMessage);
    return *NULL;
endif;

return pOutput;

end-proc;

```

6. Error monitor

Monitors for errors throughout the process, capturing exceptions and handling them effectively to maintain stable program execution and return appropriate error messages to the user.

```

/* -----
JSON error monitor
----- */
dcl-proc formatErrorMessage export;

    dcl-pi *n                pointer;
        description          varchar(256) const options(*VARSIZE);
    end-pi;

    dcl-s pOutput            pointer;
    dcl-s msg                varchar(4096);

    msg = json_message(*NULL);
    pOutput = json_newObject();
    json_setBool(pOutput : 'success' : *OFF);

```

```

    json_setStr(pOutput : 'description' :description);
    json_setStr(pOutput : 'message' :msg);

    return pOutput;

end-proc;

```

Implementing Web Services

In this example, we'll create a single program containing all four web services. While the router could easily handle separate programs for each web service, we have chosen to consolidate them into one program to streamline housekeeping and standard routines, such as initialization, navigation, and message handling.

The service program, named `corpApi2.rpg1e`, will be organized into the following sections due to its size:

1. Initialization and Message Handling
2. Service Handling
3. Screen Handling
4. Navigation

Below is a breakdown of each section:

1. Initialization and Message Handling

This section initializes the program, defines essential parameters, and manages message handling to ensure smooth execution. It captures and relays any errors encountered, allowing for effective communication of issues back to the caller.

```

<%@ pgmtype="srvpgm" pgmopt="EXPORT(*SRCFILE) srcfile(*LIBL/QSRVSRC) SRCMBR(ICECAPEX2)"
FREE="*YES" %>
<%

ctl-opt copyright('System & Method (C), 2024');
ctl-opt decEdit('0,') datEdit(*YMD.) nomain;
ctl-opt text('IcecapIntergration: Example 1');
ctl-opt bndDir('ICECAPTERM');

/include qrppleref,icecapTerm
/include qasphdr,jsonparser

```

Notice that opposite every other example there is no Main procedure. That's because the main procedure are taken over by the Router program.

```

/* -----
   Send message
   ----- */
dcl-proc sendMessage;

    dcl-pi *n                pointer;
           message          varchar(256) value;
    end-pi;

    dcl-s pOutput            pointer;

    setStatus('400'); // Bad request

    pOutput = json_newObject();
    json_setbool(pOutput : 'success' : *OFF);
    json_setstr(pOutput : 'message' : message);

    return pOutput;

end-proc;

```

2. Services Handling

Defines services to perform various operations on employee data, including retrieval, creation, updating, and deletion, allowing for efficient, background data management.

2.1 Get Employee

Retrieves specific employee data based on given criteria, providing relevant details for further processing or display.

```

/* -----
   Get employee
   ----- */
dcl-proc getEmployee export;

    dcl-pi *n                pointer;
           pInput            pointer value;
    end-pi;

```

```

dcl-s pOutput          pointer;
dcl-s pVts             pointer;
dcl-ds vts             likeds(vtsDS) based(pVts);

setContentType('application/json; charset=utf-8');
setEncodingType('*JSON');

pVts = login();
if (pVts = *NULL);
    return sendMessage('Unable to login');
endif;

navigateToCorpEmployee(pVts);
if (vtSaaTitleContains(pVts:'Work with employees'));
    // Go into employee form
    vtSetFieldNo(pVts :2 :'5'); // Field 2 is first option field
    vtSetFieldNo(pVts :3 :json_getstr(pInput :'employeeNumber'));
    vtPressFormKey(pVts: 'ENTER');
    if (not vtSaaTitleContains(pVts:'Display employee'));
        if (vts.errMsg <> '');
            pOutput = sendMessage(vts.errMsg);
        else;
            if (vtWinGetLine(pVts :27) <> '');
                pOutput = sendMessage(vtWinGetLine(pVts :27));
            else;
                pOutput = sendMessage('Unable to find screen: Display employees');
            endif;
        endif;
    else;
        pOutput = getEmployeeFormValue(pVts :pInput);
        vtPressFormKey(pVts: 'ENTER');
    endif;
    // Exit work with employess
    vtPressFormKey(pVts: 'F3');
else;
    return sendMessage('Unable to find screen: Work with employees');
endif;

vtClose(pVts);

return pOutput;

end-proc;

```

2.2 Create Employee

Adds a new employee record to the system, capturing necessary information and ensuring data integrity.

```

/* -----
   Create employee
   ----- */
dcl-proc createEmployee export;

    dcl-pi *n                pointer;
        pInput              pointer value;
    end-pi;

    dcl-s pOutput            pointer;
    dcl-s pVts              pointer;
    dcl-ds vts              likeds(vtsDS) based(pVts);
    dcl-s message            varchar(256);

    setContentType('application/json; charset=utf-8');
    setEncodingType('*JSON');

    pVts = login();
    if (pVts = *NULL);
        return sendMessage('Unable to login');
    endif;

    navigateToCorpEmployee(pVts);
    if (vtSaaTitleContains(pVts:'Work with employees'));
        // Go into employee form
        vtSetFieldNo(pVts :2 : '1'); // Field 2 is first option field
        vtSetFieldNo(pVts :3 :json_getstr(pInput : 'employeeNumber'));
        vtPressFormKey(pVts: 'ENTER');
        if (not vtSaaTitleContains(pVts:'Create employee'));
            if (vts.errMsg <> '');
                pOutput = sendMessage(vts.errMsg);
            else;
                if (vtWinGetLine(pVts :27) <> '');
                    pOutput = sendMessage(vtWinGetLine(pVts :27));
                else;
                    pOutput = sendMessage('Unable to find screen: Create employees');
                endif;
            endif;
        else;
            updateEmployeeFormValue(pVts :pInput);
            vtPressFormKey(pVts: 'ENTER');
            // Still at employee form, check for error message
            message = '';
            if (vtSaaTitleContains(pVts:'Create employee'));
                if (vts.errMsg <> '');
                    message = vts.errMsg;
                endif;
            endif;
        endif;
    endif;
end-proc;

```

```

        else;
            if (vtWinGetLine(pVts :24) <> '');
                message = vtWinGetLine(pVts :24);
            endif;
        endif;
    endif;
    if (message = '');
        pOutput = json_newObject();
        json_setBool(pOutput : 'success' : *ON);
    else;
        pOutput = sendMessage(message);
    endif;
endif;
// Exit work with employess
vtPressFormKey(pVts: 'F3');
else;
    return sendMessage('Unable to find screen: Work with employees');
endif;

vtClose(pVts);

return pOutput;

end-proc;

```

2.3 Update Employee

Modifies existing employee data as per specified updates, ensuring accurate and up-to-date employee records.

```

/* -----
Update employee
----- */
dcl-proc updateEmployee export;

    dcl-pi *n                pointer;
        pInput              pointer value;
    end-pi;

    dcl-s pOutput            pointer;
    dcl-s pVts               pointer;
    dcl-ds vts               likeds(vtsDS) based(pVts);
    dcl-s message            varchar(256);

    setContentType('application/json; charset=utf-8');
    setEncodingType('*JSON');

```

```
pVts = login();
if (pVts = *NULL);
    return sendMessage('Unable to login');
endif;

navigateToCorpEmployee(pVts);
if (vtSaaTitleContains(pVts:'Work with employees'));
    // Go into employee form
    vtSetFieldNo(pVts :2 :'2'); // Field 2 is first option field
    vtSetFieldNo(pVts :3 :json_getstr(pInput :'employeeNumber'));
    vtPressFormKey(pVts: 'ENTER');
    if (not vtSaaTitleContains(pVts:'Change employee'));
        if (vts.errMsg <> '');
            pOutput = sendMessage(vts.errMsg);
        else;
            if (vtWinGetLine(pVts :27) <> '');
                pOutput = sendMessage(vtWinGetLine(pVts :27));
            else;
                pOutput = sendMessage('Unable to find screen: Change employees');
            endif;
        endif;
    else;
        updateEmployeeFormValue(pVts :pInput);
        vtPressFormKey(pVts: 'ENTER');
        // Still at employee form, check for error message
        if (vtSaaTitleContains(pVts:'Change employee'));
            if (vts.errMsg <> '');
                message = vts.errMsg;
            else;
                if (vtWinGetLine(pVts :24) <> '');
                    message = vtWinGetLine(pVts :24);
                endif;
            endif;
            if (message = '');
                pOutput = json_newObject();
                json_setBool(pOutput :'success' :*ON);
            else;
                pOutput = sendMessage(message);
            endif;
        endif;
        // Exit work with employess
        vtPressFormKey(pVts: 'F3');
    else;
        return sendMessage('Unable to find screen: Work with employees');
    endif;

vtClose(pVts);
```

```

        return pOutput;

end-proc;

```

2.4 Delete Employee

Removes an employee record from the system, following validations to maintain data consistency.

```

/* -----
   Delete employee
   ----- */
dcl-proc deleteEmployee export;

    dcl-pi *n                pointer;
        pInput              pointer value;
    end-pi;

    dcl-s pOutput            pointer;
    dcl-s pVts              pointer;
    dcl-ds vts              likeds(vtsDS) based(pVts);
    dcl-s message            varchar(256);

    setContentType('application/json; charset=utf-8');
    setEncodingType('*JSON');

    pVts = login();
    if (pVts = *NULL);
        return sendMessage('Unable to login');
    endif;

    navigateToCorpEmployee(pVts);
    if (vtSaaTitleContains(pVts:'Work with employees'));
        // Go into employee form
        vtSetFieldNo(pVts :2 :'4'); // Field 2 is first option field
        vtSetFieldNo(pVts :3 :json_getstr(pInput :'employeeNumber'));
        vtPressFormKey(pVts: 'ENTER');
        if (not vtSaaTitleContains(pVts:'Delete employee'));
            if (vts.errMsg <> '');
                message = vts.errMsg;
            else;
                if (vtWinGetLine(pVts :27) <> '');
                    message = vtWinGetLine(pVts :27);
                endif;
            endif;
            if (message = '');
                pOutput = json_newObject();
                json_setBool(pOutput :'success' :*ON);
            endif;
        endif;
    endif;
end-proc;

```

```

        else;
            pOutput = sendMessage(message);
        endif;
    else;
        vtPressFormKey(pVts: 'ENTER');

        pOutput = json_newObject();
        json_setBool(pOutput : 'success' : *ON);
    endif;
    // Exit work with employess
    vtPressFormKey(pVts: 'F3');
else;
    return sendMessage('Unable to find screen: Work with employees');
endif;

vtClose(pVts);

return pOutput;

end-proc;

```

3. Screen Handling

Manages screen interactions, retrieving or updating form field values, and ensuring data is correctly displayed on the employee screen.

3.1 Get employee form value

Fetches a specific field value from the employee form, using either direct reference or field label, to retrieve accurate information.

```

/* -----
   Get employee form value
   ----- */
dcl-proc getEmployeeFormValue;

    dcl-pi *n                pointer;
           pVts              pointer value;
           pInput             pointer value;
    end-pi;

    dcl-s  pOutput            pointer;

    pOutput = json_newObject();

```

```
json_setstr(  
    pOutput  
    : 'name'  
    : getEmployeeFormValueByLabel(  
        pVts  
        : 'Name . . . . . :'  
    )  
);  
  
json_setstr(  
    pOutput  
    : 'middleName'  
    : getEmployeeFormValueByLabel(  
        pVts  
        : 'Midle Name . . . . . :'  
    )  
);  
  
json_setstr(  
    pOutput  
    : 'surName'  
    : getEmployeeFormValueByLabel(  
        pVts  
        : 'Surname . . . . . :'  
    )  
);  
  
json_setstr(  
    pOutput  
    : 'department'  
    : getEmployeeFormValueByLabel(  
        pVts  
        : 'Department . . . . . :'  
    )  
);  
  
json_setstr(  
    pOutput  
    : 'phoneNo'  
    : getEmployeeFormValueByLabel(  
        pVts  
        : 'Phone No. . . . . :'  
    )  
);  
  
json_setstr(  
    pOutput  
    : 'birthday'  
    : getEmployeeFormValueByLabel(  
        pVts
```

```
        : 'Birthday . . . . . :'  
    )  
);  
  
json_setstr(  
    pOutput  
    : 'hired'  
    : getEmployeeFormValueByLabel(  
        pVts  
        : 'Hired . . . . . :'  
    )  
);  
  
json_setstr(  
    pOutput  
    : 'job'  
    : getEmployeeFormValueByLabel(  
        pVts  
        : 'Job . . . . . :'  
    )  
);  
  
json_setstr(  
    pOutput  
    : 'educationLevel'  
    : getEmployeeFormValueByLabel(  
        pVts  
        : 'Education Level . . . :'  
    )  
);  
  
json_setstr(  
    pOutput  
    : 'gender'  
    : getEmployeeFormValueByLabel(  
        pVts  
        : 'Gender . . . . . :'  
    )  
);  
  
json_setstr(  
    pOutput  
    : 'salary'  
    : getEmployeeFormValueByLabel(  
        pVts  
        : 'Salary . . . . . :'  
    )  
);  
  
json_setstr(  

```

```

        pOutput
        : 'bonus'
        : getEmployeeFormValueByLabel(
            pVts
            : 'Bonus . . . . . : '
        )
    );

    json_setstr(
        pOutput
        : 'commission'
        : getEmployeeFormValueByLabel(
            pVts
            : 'Commission . . . . . : '
        )
    );

    return pOutput;

end-proc;

```

The unified get mechanism per field have its own procedure called: `getEmployeeFormValueByLabel`.

```

/* -----
   Get employee form value
   ----- */
dcl-proc getEmployeeFormValueByLabel;

    dcl-pi *n                varchar(256);
        pVts                pointer;
        label               varchar(256) value;
    end-pi;

    dcl-ds vtField           likeds(vtFieldDS);
    dcl-ds location          likeds(vtLocationDS);

    location = vtLocate(
        pVts
        :VT_OUTPUT
        :vtLinCol(1:1)
        :VT_EAST
        :label
    );
    if (location.lin > 0);
        vtGetField(
            pVts
            :location
            :vtField

```

```

    );
    return %trim(vtField.value);
endif;

return '';

end-proc;

```

3.2 Update employee form value

Updates a specific field on the employee form, using a direct field reference or label, to ensure precise data modification on-screen.

```

/* -----
   Update employee form value
   ----- */
dcl-proc updateEmployeeFormValue;

    dcl-pi *n;
        pVts                pointer value;
        pInput              pointer value;
    end-pi;

    if (json_has(pInput : 'name'));
        setEmployeeFormValueByLabel(
            pVts
            : 'Name . . . . . '
            : json_getstr(pInput : 'name')
        );
    endif;

    if (json_has(pInput : 'middleName'));
        setEmployeeFormValueByLabel(
            pVts
            : 'Middle Name . . . . . '
            : json_getstr(pInput : 'middleName')
        );
    endif;

    if (json_has(pInput : 'surName'));
        setEmployeeFormValueByLabel(
            pVts
            : 'Surname . . . . . '
            : json_getstr(pInput : 'surName')
        );
    endif;

```

```
if (json_has(pInput : 'department'));
    setEmployeeFormValueByLabel(
        pVts
        : 'Department . . . . . '
        : json_getstr(pInput : 'department')
    );
endif;

if (json_has(pInput : 'phoneNo'));
    setEmployeeFormValueByLabel(
        pVts
        : 'Phone No. . . . . '
        : json_getstr(pInput : 'phoneNo')
    );
endif;

if (json_has(pInput : 'birthday'));
    setEmployeeFormValueByLabel(
        pVts
        : 'Birthday . . . . . '
        : json_getstr(pInput : 'birthday')
    );
endif;

if (json_has(pInput : 'hired'));
    setEmployeeFormValueByLabel(
        pVts
        : 'Hired . . . . . '
        : json_getstr(pInput : 'hired')
    );
endif;

if (json_has(pInput : 'job'));
    setEmployeeFormValueByLabel(
        pVts
        : 'Job . . . . . '
        : json_getstr(pInput : 'job')
    );
endif;

if (json_has(pInput : 'educationLevel'));
    setEmployeeFormValueByLabel(
        pVts
        : 'Education Level . . . . '
        : json_getstr(pInput : 'educationLevel')
    );
endif;

if (json_has(pInput : 'gender'));
```

```

        setEmployeeFormValueByLabel(
            pVts
            : 'Gender . . . . . '
            :json_getstr(pInput : 'gender')
        );
    endif;

    if (json_has(pInput : 'salary'));
        setEmployeeFormValueByLabel(
            pVts
            : 'Salary . . . . . '
            :json_getstr(pInput : 'salary')
        );
    endif;

    if (json_has(pInput : 'bonus'));
        setEmployeeFormValueByLabel(
            pVts
            : 'Bonus . . . . . '
            :json_getstr(pInput : 'bonus')
        );
    endif;

    if (json_has(pInput : 'commission'));
        setEmployeeFormValueByLabel(
            pVts
            : 'Commission . . . . . '
            :json_getstr(pInput : 'commission')
        );
    endif;

    return;

end-proc;

```

The unified set mechanism per field have its own procedure called: **setEmployeeFormValueByLabel**

```

/* -----
   Set employee form value
   ----- */
dcl-proc setEmployeeFormValueByLabel;

    dcl-pi *n;
        pVts                pointer;
        label                varchar(256) value;
        value                varchar(256) value;
    end-pi;

```

```

dcl-ds location          likeds(vtLocationDS);

location = vtLocate(
    pVts
    :VT_INPUT
    :vtLinCol(1:1)
    :VT_EAST
    :label
);
if (location.lin > 0);
    vtSetField(
        pVts
        :location.lin
        :location.col
        :value
    );
endif;

return;

end-proc;

```

4. Navigation

Controls the navigation flow within the system, handling transitions to specific screens and performing initial logins to reach the main interface.

4.1 Navigate to Corp. employee screen

Directs the user to the corporate employee screen, initiating the relevant screen for employee management tasks.

```

/* -----
   Navigate to Corp. employee screen
   ----- */
dcl-proc navigateToCorpEmployee;

    dcl-pi *n;
        pVts          pointer;
    end-pi;

    dcl-ds vtLogon      likeds(vtLogonDS);
    dcl-s  screenFound  ind;

    screenFound = *OFF;

```

```

    dou (screenFound);
    select;
    when (vtSaaTitleContains(pVts:'Command Entry'));
        vtSetFieldNo(pVts: 1 : 'go corppgm/corpmnu');
        vtPressKey(pVts: vtENTER);
    when (vtSaaTitleContains(pVts:'Corp. Data Main Menu'));
        vtSetFieldNo(pVts: 1 : '1');
        vtPressKey(pVts: vtENTER);
    when (vtSaaTitleContains(pVts:'Work with employees'));
        screenFound = *ON;
    ends1;
enddo;

end-proc;

```

4.2 Start Virtual terminal, login and make sure to end on main screen

Launches a virtual terminal session, handles the login process, and confirms arrival on the main screen for seamless operation initiation.

```

/* -----
   Start Virtual terminal, login and make sure to end on main screen
   ----- */
dcl-proc login;

    dcl-pi *n                pointer;
    end-pi;

    dcl-s pVts                pointer;
    dcl-ds vts                likeds(vtsDS) based(pVts);
    dcl-ds vtLogon            likeds(vtLogonDS);
    dcl-s screenFound        ind;

    // Open IceCap Virtual Terminal Session
    vtLogon = *LOVAL;
    vtLogon.keepInput = *ON;
    pVts = vtOpen(vtLogon);

    screenFound = *OFF;
    dou (screenFound);
    select;
    // Logon if the signon is displayed
    when (vtSaaTitleContains(pVts:'Sign On'));
        vtSetFieldNo(pVts: 1 : 'DEMO');    // First field is the userprofile
        vtSetFieldNo(pVts: 2 : 'DEMO');    // Second field is the password
        vtSetFieldNo(pVts: 3 : 'QCMD');    // Program to run - QCMD

```

```

        vtPressKey(pVts: vtENTER);
        // Check message
        if (vts.errMsg <> '');
            sendMessage(vts.errMsg);
            vtClose(pVts);
            return *NULL;
        endif;
        if (vtWinGetLine(pVts :24) <> ''
        and %subst(vtWinGetLine(pVts :24) :1 :3) = 'CPF');
            sendMessage(vtWinGetLine(pVts :24));
            vtClose(pVts);
            return *NULL;
        endif;
        when (vtSaaTitleContains(pVts:'LOG P'));
            if (vtWinGetLine(pVts :3) <> ''
            and %subst(vtWinGetLine(pVts :3) :2 :21) <> 'Logget p' sidste gang');
                sendMessage(vtWinGetLine(pVts :3));
                vtClose(pVts);
                return *NULL;
            else;
                vtPressKey(pVts: vtENTER);
            endif;
        when (vtSaaTitleContains(pVts:'Attempt to Recover Interactive Job'));
            vtSetFieldNo(pVts: 1 : '90');
            vtPressKey(pVts: vtENTER);
        when (vtSaaTitleContains(pVts:'Display Program Messages')
        or    vtSaaTitleContains(pVts:'VIS PROGRAMMEDDELELSER'));
            vtPressKey(pVts: vtENTER);
        when (vtSaaTitleContains(pVts:'Command Entry'));
            screenFound = *ON;
        endl;
    enddo;

    return pVts;

end-proc;

```

The new service layer is now ready for testing and can be accessed accordingly to the below examples. To do so you need to install a product like Postman (<https://www.postman.com/>) on your computer to use the RESTful services correct.

Read:

(GET) <http://myIBMi:7700/corporate/employee/000010>

Response:

```
{
  "name": "JOHN",
  "middleName": "I",
  "surName": "HAAS",
  "department": "A00",
  "phoneNo": "3425",
  "birthday": "12-01-1965",
  "hired": "23-08-1995",
  "job": "ANALYST",
  "educationLevel": "13",
  "gender": "M",
  "salary": "52750,00",
  "bonus": "1100,00",
  "commission": "1400,00"
}
```

Create:

(POST) <http://myIBMi:7700/corporate/employee>

```
{
  "employeeNumber": "000011",
  "name": "WILL",
  "middleName": "",
  "surName": "SMITH",
  "department": "A00",
  "phoneNo": "1234",
  "birthday": "01-01-1960",
  "hired": "01-01-2000",
  "job": "ANALYST",
  "educationLevel": "13",
  "gender": "M",
  "salary": "43210,00",
  "bonus": "1230,00",
  "commission": "987,00"
}
```

Response:

```
{
  "success": true
}
```

```
}
```

Update request:

```
(PUT) http://myIBMi:7700/corporate/employee/000011
```

```
{  
  "name": "WILL",  
  "middleName": "",  
  "surName": "SMITH",  
  "department": "B01",  
  "phoneNo": "4321",  
  "birthday": "01-01-1960",  
  "hired": "01-01-2000",  
  "job": "MANAGER",  
  "educationLevel": "13",  
  "gender": "M",  
  "salary": "54320,00",  
  "bonus": "2340,00",  
  "commission": "1234,00"  
}
```

Response:

```
{  
  "success": true  
}
```

Delete request:

```
(DELETE) http://myIBMi:7700/corporate/employee/000011
```

Response:

```
{  
  "success": true  
}
```

Customized Workflows

By breaking down the ERP system into smaller web services, you can more easily integrate these RESTful services into new workflows. This modular approach enhances flexibility and adaptability, allowing each service to be updated or replaced independently without impacting the entire system. With a service layer, development teams can work on different services simultaneously, accelerating the overall process and enabling quicker deployment of new features or updates.

Effectively implementing a service layer "blackboxes" the ERP system, encapsulating the legacy system and eliminating the need for changes to core programs, as well as the need for RPG/Cobol programmers. In summary, adopting a web services architecture for an ERP system enhances modularity and adaptability, streamlines development and management, and makes the system more robust and easier to evolve.

Is IceCap the Only Option?

The 5250 Employee program discussed in this chapter is an exceptionally simple example with minimal business logic. While it serves well as an illustration, there are alternatives to transitioning to web services using IceCap. For straightforward programs like this, consider converting them into full web applications using Sitemule Architect or to web services with IceBreak Enterprise.

Virtual Terminal

The Virtual Terminal component in IceCap comprises two main elements: (1) the Virtual Terminal Session (VTS) and (2) a suite of Virtual Terminal Commands (API) that interact with the active session.

The Virtual Terminal Session (VTS)

The **Virtual Terminal Session (VTS)** in IceCap functions as a core component that emulates and manages 5250 terminal sessions on IBM i systems. The VTS acts as an interface between the traditional terminal screen data and the modernized web interface provided by IceCap, allowing for automated workflows, data manipulation, and screen interaction without altering the underlying IBM i applications.

Role and Importance of VTS

1. Session Management:

- **Open and Close Sessions:** The VTS handles the initiation (`vtOpen`) and termination (`vtClose`) of virtual terminal sessions. Each session maintains its own state, screen buffer, and cursor position, allowing multiple screens to be managed independently.
- **Resource Allocation:** By encapsulating each 5250 interaction as a session, VTS ensures efficient resource management and avoids conflicts across different terminal tasks.

2. Screen Interaction and Automation:

- **Screen Data Handling:** The VTS captures screen content, manages the screen buffer, and supports field-level data retrieval and entry. Commands like `vtGetField` or `vtSetField` allow developers to read or write data in specific fields programmatically.
- **Cursor Positioning and Navigation:** VTS commands position the cursor (`vtSetCursor`) and simulate key presses (e.g., "Enter", F-keys) to navigate screens, enabling workflows to proceed automatically as if they were manually operated.

3. Data Retrieval and Manipulation:

- **Field Locating:** Using the VTS, developers can search for specific fields by label or keyword using commands like `vtLocate`. This ability to locate and interact with fields is crucial for capturing data or filling out forms programmatically.
- **Screen Capturing:** The VTS can capture entire screens (`vtGetScreen`) and convert data into JSON for web-based processing, making it easier to integrate legacy IBM i data with modern applications.

4. Enhanced User Interface:

- **Transformation to Web Formats:** The VTS translates traditional 5250 screen formats into web-compatible data, allowing IBM i programs to run as web applications without changing their underlying code. It supports the addition of interactive elements (e.g., dropdowns, date pickers) through integration with IceCap's Tag and Plugin features.
- **Improved Workflow and User Experience:** By handling terminal sessions in the background, the VTS allows users to benefit from a web-based, multitasking environment that is more aligned with modern UI expectations, while retaining the stability and functionality of IBM i.

The Virtual Terminal Session (VTS) in IceCap is vital for bridging the gap between IBM i's legacy 5250 interface and a modernized, web-based experience. It manages screen sessions, automates data entry and navigation, supports data retrieval, and enhances the presentation of IBM i applications without requiring changes to the core application code.

Virtual Terminal (VT) Commands

VT Commands are a powerful toolset (API) that allows for programmatic control over the terminal session. Through these commands, users can automate interactions within the session, navigate screens, and perform data entry and retrieval tasks. This capability is essential for organizations that need to streamline workflows across legacy IBM i applications. By using VT Commands, repetitive tasks are automated, improving efficiency and reducing reliance on manual processes in legacy systems.

Core Functions of VT Commands

The primary uses of VT commands include:

1. Automating Data Entry and Retrieval:

VT commands such as `vtSetFieldNo` and `vtGetFieldNo` allow specific fields to be set or read by their field numbers. This function is particularly beneficial for scenarios where the same data must be input across multiple sessions or when retrieving data from specific fields for validation or reporting.

2. Screen Navigation and Process Flow:

Commands like `vtPressKey` simulate key presses (e.g., "Enter," "Page-Up," "F3") to navigate through screens automatically. This ability allows applications to control workflows and move through screens as a user would manually, creating smoother, automated transitions.

3. Enhancing Data Handling and Validation:

Commands such as `vtGetFieldNo` retrieve not just the data from a field but also metadata like field attributes and format. This allows applications to validate user entries or to ensure the data is accurately formatted before processing, reducing the risk of errors.

4. Improving Workflow Efficiency:

VT commands can automate complex workflows that involve multiple 5250 screens. This is particularly useful in high-frequency tasks, such as entering data into multiple fields across different screens in a sequence, as it reduces the time required to complete these tasks and minimizes user intervention.

Use Case #1: Automated Employee Search and Data Validation

Objective: A company uses a 5250 terminal application for HR data management. To streamline an employee search and data entry workflow, they implement VT commands to automate the process.

Steps in the Workflow:

1. Login Automation:

- Use `vtSetFieldNo` to set the username and password in the login fields.
- Use `vtPressKey(vtENTER)` to submit the login.

2. Employee Search:

- Use `vtSetFieldNo` to enter the employee ID in the search field.
- Submit the search using `vtPressKey(vtENTER)` to navigate to the results screen.
- If the screen displays multiple pages of results, use `vtPressKey(vtPAGE_DOWN)` to scroll through pages.

3. Retrieve Employee Data:

- Use `vtGetFieldNo` to capture and validate fields like employee name and department.
- This data retrieval helps verify that the correct record was selected, allowing the automation script to make decisions based on field content (e.g., proceeding only if the department matches a specific value).

4. Field Modification for Reporting:

- Use `vtSetMetaData` to adjust fields to a specific type (e.g., combo box for department selection) based on retrieved employee data.

Code Example

Here's a code example illustrating how VT commands streamline the workflow:

```
// Step 1: Set Login credentials and submit
vtSetFieldNo(pVts: 1 : 'USER123');    // Sets User ID in field 1
vtSetFieldNo(pVts: 2 : 'PASS123');    // Sets Password in field 2
vtPressKey(pVts: vtENTER);            // Press "Enter" to Log in

// Step 2: Search for employee by ID
vtSetFieldNo(pVts: 3 : 'EMP456');      // Sets employee ID in search field
vtPressKey(pVts: vtENTER);            // Submits search

// Step 3: Retrieve employee name for validation
vtGetFieldNo(pVts: 4 : vtFieldDS);     // Retrieves data from field 4 (Employee Name)
if (vtFieldDS.value = 'John Doe');     // Checks if the correct employee is selected

// Step 4: Modify field to a combo box if necessary
vtSetMetaData(pVts: 5 : '{"xtype":"comboBox"}');
// Sets a field as a combo box for department
endif;
```

Use Case: Detailed Functionality of a VT command

Commands like `vtPressKey` facilitate seamless navigation across screens, enabling multi-screen workflows without manual interaction. It allows developers to simulate key presses programmatically, which is especially useful in automating workflows in 5250 terminal applications. This function works by intercepting and transmitting specific keystrokes, replicating the action a user would perform when interacting with a 5250 screen, such as pressing "Enter," "Page-Up," "F3," or custom function keys.

1. Simulating User Interaction

The `vtPressKey` function initiates a command that resembles a user pressing a key on the physical terminal. This command sends both the keypress and any entered data back to the session server, triggering the next action in the workflow.

2. Automated Workflow

IceCap applications often require users to input data across multiple screens, where specific keys trigger certain actions or navigate between screens. By programmatically pressing these keys, `vtPressKey` enables developers to automate complex workflows and repetitive tasks. For example, pressing "F12" might navigate to a previous screen, or "Enter" might submit data.

3. Efficient Screen Transitions

The function can handle key presses that lead to multiple screen transitions in sequence. If combined with other API calls (e.g., `vtSetFieldNo` to enter data in fields), it helps complete an end-to-end process without manual intervention.

4. Custom Keypresses

`vtPressKey` is flexible and allows a range of key values:

- **Standard Keys:** `vtENTER`, `vtF3`, `vtF12`, etc., for navigation and function-specific commands.
- **Control Keys:** Keys like `Page-Up` or `Page-Down` for navigating data-rich screens.
- **System-Specific Keys:** If an application uses custom key assignments, `vtPressKey` can support those as well.

Practical Use Case

Consider an HR system on a 5250 terminal where user information needs to be retrieved across different screens:

- **Step 1:** Open the "Employee Search" screen.
- **Step 2:** Enter the employee ID in the search field.
- **Step 3:** Press "Enter" to submit the search.

- **Step 4:** If results show multiple records, use **Page-Down** to scroll.

Using **vtPressKey** to automate this:

```
vtSetFieldNo(pVts: 5 : 'EMP123'); // Set the Employee ID in field 5
vtPressKey(pVts: vtENTER);        // Submit the search with "Enter"

// After displaying results
vtPressKey(pVts: vtPAGE_DOWN); // Scrolls down if there are multiple pages
```

In this example:

- **Efficiency Gains:** Automates a repetitive process, saving time and reducing manual data entry errors.
- **Screen Navigation:** Allows seamless movement through the application without user intervention.
- **Error Minimization:** Reduces the risk of pressing incorrect keys manually.

Technical Considerations

1. **Error Handling:** When using **vtPressKey**, ensure that the screen transitions as expected after each press. If a key press leads to unexpected behavior (e.g., error messages), additional error handling routines may be necessary.
2. **Synchronization with Other APIs:** The function works well with **vtSetField** and **vtGetScreen** for a complete data-entry and screen-navigation process.
3. **Response Time:** Each use of **vtPressKey** involves a roundtrip to the server, which can introduce slight delays if used in rapid succession. Testing the workflow under load conditions can ensure it performs as required.

Benefits of Using VT Commands

1. **Reduced Manual Entry:** VT commands automate data entry and retrieval, decreasing the time users spend on repetitive tasks.
2. **Increased Accuracy:** By programmatically setting and validating fields, VT commands reduce the risk of manual errors.
3. **Streamlined Navigation:** Commands like **vtPressKey** facilitate seamless navigation across screens, enabling multi-screen workflows without manual interaction.

4. **Enhanced User Experience:** Automations reduce screen-switching, speeding up complex processes and making user interactions more efficient.

The VT commands in IceCap are invaluable for automating and optimizing workflows within 5250 terminal applications. They provide a robust toolkit for manipulating fields, navigating screens, and enhancing data handling without altering the core application. These capabilities enable organizations to maintain and modernize legacy applications effectively, improving user productivity and data accuracy while minimizing manual intervention.

Status bar

The Status Bar in IceCap is designed to improve the user experience and productivity by providing real-time updates on the current session and application state. Positioned at the bottom of the application window, it gives users immediate access to essential session details and operational features.

Status Bar Features

Feature	Description
Position	Shows the user's current position on the 5250 screen, helping them navigate efficiently within sessions.
Copy	Allows users to capture the screen content quickly. CTRL-C copies as plain text, and CTRL-E copies in a tabular format for use in spreadsheets like Excel. Accumulated screenshots can be copied with CTRL-SHIFT-C and CTRL-SHIFT-E for multiple entries.
Print	Creates a screenshot of the current tab. Users can select different formats and choose to include or exclude graphical elements, such as background images.
Language	Displays the active language setting. Users can confirm or switch languages as needed, supporting multilingual environments. For details on this feature, refer to the Translation chapter.
Translation	Enables toggling of Translation Mode, allowing users to manage translations on-the-fly. More information is in the Translation chapter.
Mode	Indicates the current display mode in IceCap, allowing users to switch seamlessly between modes for different display preferences.

Usage Notes

- **Deactivation in Flip Mode:** The Status Bar is turned off by default in IceCap's Flip Mode. To enable it, adjust the configuration setting to `showStatusBar="yes"`.

The Status Bar is a central component in the IceCap interface, designed to support efficient navigation, accurate data handling, and adaptable language settings. By centralizing these tools, it enhances overall usability and productivity, making it an invaluable feature for users working within IBM i environments.

Translation

The Translation feature in IceCap allows you to customize your IBM i applications to support multiple languages, providing a more accessible experience for a global user base. With this functionality, you can seamlessly translate or modify field labels, menu options, function keys (F-keys), messages, and other system elements. This ensures that users can interact with the application in their preferred language while also improving the clarity and relevance of field labels.

IceCap offers the flexibility to apply translations either on individual screens or system-wide, depending on your requirements. This adaptability enhances the onboarding process for international users, improves usability, and ensures clear and consistent communication across different systems.

By implementing translations in IceCap, you foster a more inclusive, user-friendly environment, significantly improving the overall user experience and increasing productivity.

How Does Translation Work?

The translation feature in IceCap is designed to be user-friendly and accessible to any user.

1. Selecting Language

In the status bar in the lower right corner of the screen, you can see the current language of the page, indicated by the international two-letter code (e.g., "en" for English, "da" for Danish). Clicking on the language code brings up a menu with the available languages. When you choose a different language, the screen texts change immediately. Texts that have not yet been translated will remain in the original language.

2. Activating and using Translation

Clicking the "globe" icon next to the language code allows users to correct or add translations. This globe icon is only visible if the user has the "Administrator" role in Portfolio. Translation is only possible when IceCap is in Flat mode, where the text position and length are fixed. Activating translation in other modes will switch the screen to Flat mode. When you move the cursor over a text area, such as a field label, the affected area will be highlighted. A mouse click opens the text for correction in the current language, showing the corresponding flag. The original text is displayed if it

needs to be changed or translated. Click on the flag to translate the selected text into other languages. You can recall the original text by clicking on the "globe" in the list of languages.

3. Choosing Global or Local Translation

By default, translations are done per screen, which is useful for clarifying the specific use of a field in the current process. Many field labels, options, and function keys are consistent system-wide. Selecting "Global" in the text area selection menu will apply the translation globally across all 5250 screens, ensuring consistency in naming and definitions.

Correcting or Changing Misleading Text and Field Labels

The translation feature in IceCap can also be leveraged to correct or clarify misleading screen text, option descriptions, or field labels. In many cases, users may attempt to input information that wasn't originally anticipated during the program's design. As a result, they might repurpose fields that are no longer relevant or were previously unused, leading to confusion and reliance on informal knowledge due to insufficient documentation.

IceCap's translation feature allows you to address this issue by updating and correcting field labels and descriptions, making them more accurate and informative. This ensures that users interact with the system more efficiently, reducing errors and improving overall clarity.

Understanding the Technique Behind Translation

The Translation component scans the data stream from the 5250 screen on its way to the web client. The original text is compared with translations. If a translation is marked as "Global," the text is immediately replaced with the translation for the current language. For texts not marked as "Global," the screen title and text area position are used to identify a "Local" translation. When choosing between global and local translation, the precision of the translation must be considered.

In the `<default>` section of the configuration file `IceCap_Config.xml`, you can activate or deactivate the translation feature and revise the list of available languages:

```
<defaults
```

```
useScreenTranslation="yes"  
translateToLanguages="da,en,de"  
</>
```

Please note that deactivating the translation feature will not remove the corresponding icons from the status bar.

Start PC Organizer

The "Start PC Organizer" feature on IBM i systems bridges the gap between traditional 5250 programs and modern PC-based applications. For the past 30 years, it has been the primary method for enabling direct command execution and integration between these two environments. This feature has been widely used in IBM i installations, providing a robust solution for complex workflows that require the capabilities of both IBM i and PC platforms.

IceCap and Portfolio seamlessly support this method without any changes to the setup.

How Does Start PC Organizer Work?

The "Start PC Organizer" (STRPCO) command allows users to execute PC commands or launch PC programs directly from their 5250 sessions, enhancing integration between IBM i and PC environments. Here's a brief overview of how it works:

1. Initialization

To begin using the PC Organizer, the **STRPCO** command must be executed. This command initiates a session that allows PC commands to be sent from the IBM i system to the connected PC. The basic syntax is:

```
STRPCO
```

2. Sending PC Commands

Once the PC Organizer is started, users can send specific commands to the PC using the **STRPCCMD** command. This command allows users to execute any PC command or program accessible from the PC. For example, to open a specific document or application, the syntax would be:

```
STRPCCMD PCCMD('notepad.exe myfile.txt')
```

This command instructs the PC to open Notepad and load myfile.txt.

3. System Communication

The PC Organizer facilitates communication between IBM i applications and PC applications, useful for tasks that require data sharing or joint operations. Data generated on IBM i can be easily transferred to a PC application for further processing or presentation. To enable this communication, products like “IBM i Access Client Solutions” need to be installed on the user’s PC.

4. Workflow Integration

Integrating PC commands into IBM i workflows can significantly enhance productivity. Users can automate the launching of PC applications based on specific conditions or events within the 5250 programs. By combining `STRPCO` and `STRPCCMD` commands with 5250 programming, sophisticated workflows can be developed that leverage the strengths of both environments.

How Does Start PC Organizer Work in IceCap?

IceCap does not depend on the installation of “IBM i Access Client Solutions” because the Portfolio Client includes a service that facilitates communication similarly to previous methods. IceCap will try to open files and documents in a tab inside Portfolio using a browser-based view whenever possible. If not supported, IceCap will open the file by activating the relevant application on the user’s PC.

Interception and Adjusting the Communication

When configuring IceCap, it is possible to customize the communication between IBM i and the PC application. This is done by changing or replacing the template for the PCO Program component. Look for the keyword `pcoProgram` in the `<breakOut>` section of the `IceCap-config.xml` file.

Example configuration in XML:

```
<breakOut
  navigatorProgram="*LIBL/*NONE"
  emulatorProgram="*LIBL/*NONE"
  interpretationProgram="*LIBL/*NONE"
  pcoProgram="*LIBL/PCOSERVER"
  i18nXlateProgram="*LIBL/*NONE"
  xxx_deviceNameProgram="*LIBL/*NONE"
  xxx_varyoffExitProgram="*LIBL/*NONE"
/>
```

Adjustments could include extended security, additional business logic, changes to 5250 programs that can't be recompiled, and codepage issues.

Keystroke Buffering

IceCap is designed to accommodate fast-typing users by utilizing an efficient keystroke buffering system. This feature ensures that even when users type faster than the system can display screens, their input is not lost. The buffering mechanism collects the keystrokes, including command and function keys (e.g., Enter, PageUp/Down, F1–F24), and processes them as the system catches up, guaranteeing seamless input across multiple screens.

How Keystroke Buffering Works

1. **Input Collection:** As the user types, IceCap collects all keystrokes into an internal buffer. This buffer temporarily stores the input until the system is ready to display the next screen.
2. **Input Retention:** The buffered keystrokes remain in the buffer even if the screen is delayed. When the next screen or input field is ready to process the user input, IceCap retrieves the buffered keystrokes and applies them automatically. This ensures that no keystrokes are lost, even during rapid typing.
3. **Key Handling:** The buffering system not only retains alphanumeric input but also captures shortcuts, command and function key presses. These key inputs are similarly stored in the buffer and applied when the corresponding screen appears. This prevents disruptions in workflows that rely on the use of these keys.
4. **Sequential Processing:** The system processes the buffered keystrokes sequentially. Once the next screen is rendered, the buffer is emptied in the order the keys were pressed. This guarantees that the user's actions, including commands and function keys, are executed in the correct sequence.
5. **Real-time Input Display:** As screens become available, IceCap will display the user's input as if it were typed in real-time, allowing the system to catch up without requiring the user to retype commands or inputs.

Key Benefits

- **Improved User Experience:** Users can type naturally, even at high speeds, without needing to worry about system performance delays. The buffer ensures that all input is registered and applied correctly.

- **No Lost Input:** Keystrokes won't disappear during periods of screen delays, ensuring that all commands and input data are properly submitted.
 - **Support for all Keys:** Shortcuts, Command and Function key inputs are buffered just like standard keystrokes, so workflows involving commands like CTRL+A, Enter, PageUp or F3 are unaffected by system lag.
-

Best Practices

- For optimal use of keystroke buffering, ensure that the system's processing speed matches user requirements. Although buffering compensates for brief delays, prolonged system issues may still affect performance.

Demo Environment

The Demo Environment in IceCap serves as a valuable resource for users to explore and understand the full range of IceCap's capabilities. It provides a fully functional example setup that demonstrates how IceCap can transform traditional 5250 applications into modern, web-based interfaces.

By using the Demo Environment, users can experience firsthand the enhanced usability, improved workflows, and advanced features that IceCap offers, such as visual aids, automated data entry, and streamlined navigation. This environment serves as a practical guide for learning how to set up and customize IceCap for specific business needs, providing a comprehensive template for real-world applications.

Installation

The Demo Environment is optional to install alongside the other IceCap and Portfolio products. If it has not yet been installed, it can be added later by re-running the installation procedure and selecting only the demo environment.

The Demo Environment consists of the following three libraries:

Library	Description
CORPDATA	Contains demo data originally created by IBM as a sample database for educational purposes. This database is still referenced in the Db2 for IBM i SQL Reference . IceCap uses this data for its examples in the manual and Demo Environment.
CORPPGM	Holds RPG programs created as examples of traditional 5250 programs that IceCap can transform. A menu is also available within this library, accessible via the command: GO CORPPGM/CORPMNU.
CORPDMD	Contains the definition files for Sitemule Model Studio and several SQL views used in conjunction with web applications defined by Model Studio.

Refreshing Sample Tables

The tables in the CORPDATA library can be refreshed if needed by following the instructions provided at [Sample Tables](#). To refresh the data, a stored procedure is available as part of the system. It includes DDL statements to create and populate the sample tables. The procedure can be executed from any SQL interface by issuing the following command, where **CORPDATA** is the desired schema name:

```
CALL QSYS.CREATE_SQL_SAMPLE('CORPDATA')
```

Ensure the schema name is specified in uppercase, and the schema must not already exist.

Uninstalling

To uninstall the Demo Environment, simply delete the three libraries mentioned above (CORPDATA, CORPPGM, and CORPDMD) and remove the associated applications from the Portfolio menu system.

Resources

Database

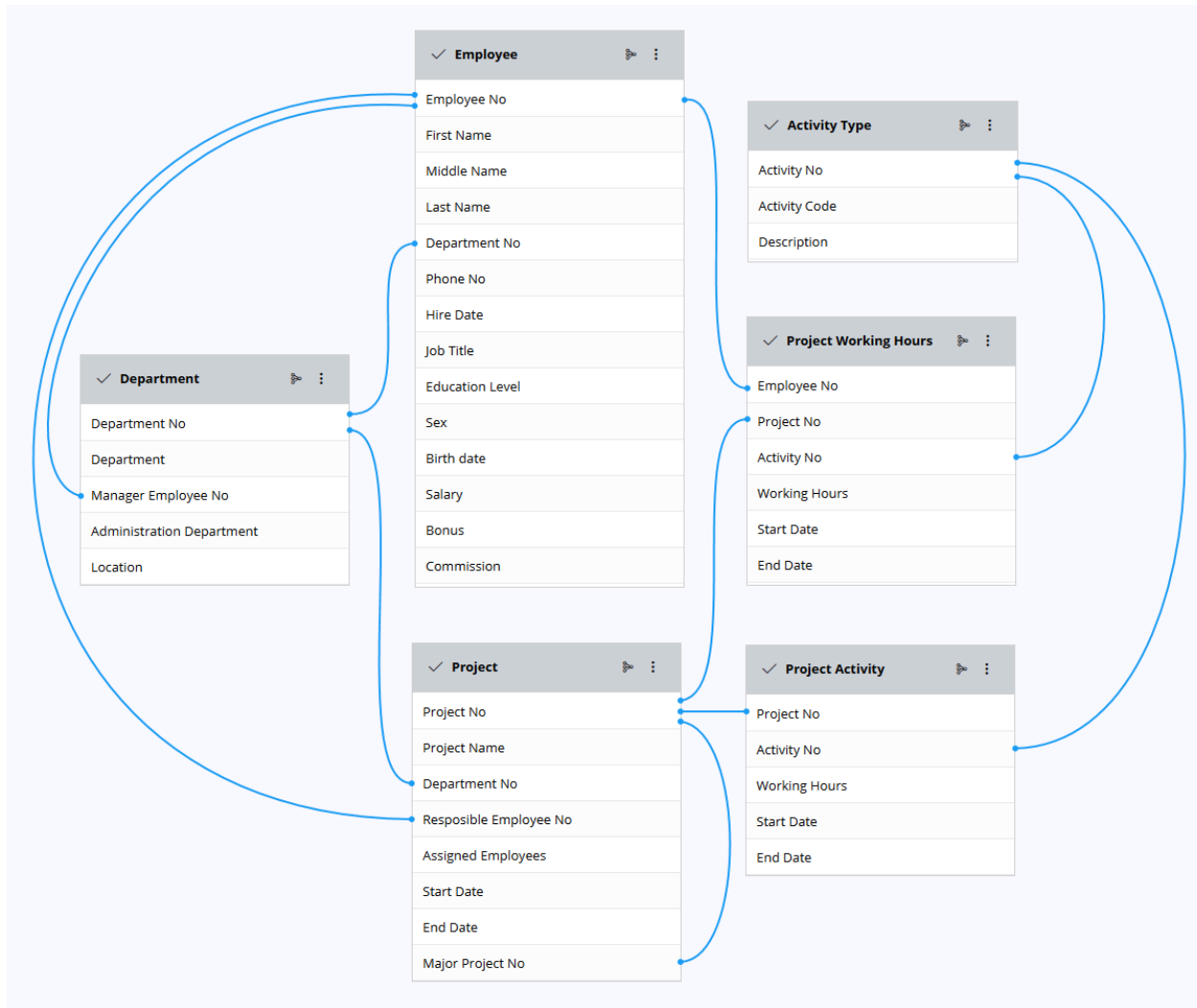
The demo environment utilizes the following key tables:

Object	Type	Library	Attribute	Description
ACT	*FILE	CORPDATA	PF	Activity Type
DEPARTMENT	*FILE	CORPDATA	PF	Department
EMPLOYEE	*FILE	CORPDATA	PF	Employee
EMPPROJECT	*FILE	CORPDATA	PF	Employee
PROJECT	*FILE	CORPDATA	PF	Project Activity
PROJECT	*FILE	CORPDATA	PF	Project

These tables are linked together in the relational database, forming the foundation of the demo application.

Database Diagram

The following diagram shows how the involved files are related within the database structure. This diagram is originally from Sitemule Model Studio, which is not part of IceCap but provides a useful overview of the database design.



Programs

The RPG programs included in the Demo Environment were generated using a tool that ensures a consistent interface and behavior across all programs. They were designed following the IBM Systems Application Architecture (SAA) Standard from 1987, ensuring a realistic representation of programs from that era.

For a detailed overview of the available programs and related objects, including their sources, refer to the following list:

Object	Type	Library	Attribute	Description
ACTIVIC	*PGM	CORPPGM	CLLE	Activity Codes
ACTIVIRA	*PGM	CORPPGM	RPGLE	Activity Code List
ACTIVIDA	*FILE	CORPPGM	DSPF	Activity Code List
ACTIVIRB	*PGM	CORPPGM	RPGLE	Activity Code Form
ACTIVIDB	*FILE	CORPPGM	DSPF	Activity Code Form
DEPARTC	*PGM	CORPPGM	CLLE	Departments
DEPARTRA	*PGM	CORPPGM	RPGLE	Department List
DEPARTDA	*FILE	CORPPGM	DSPF	Department List
DEPARTRB	*PGM	CORPPGM	RPGLE	Department Form
DEPARTDB	*FILE	CORPPGM	DSPF	Department Form
EMPLOYC	*PGM	CORPPGM	CLLE	Employees
EMPLOYRA	*PGM	CORPPGM	RPGLE	Employee List
EMPLOYDA	*FILE	CORPPGM	DSPF	Employee List
EMPLOYRB	*PGM	CORPPGM	RPGLE	Employee Form
EMPLOYDB	*FILE	CORPPGM	DSPF	Employee Form
EMPLOYRC	*PGM	CORPPGM	RPGLE	Employee Form, Parm(Action, Employee no)
PROJCTC	*PGM	CORPPGM	CLLE	Projects
PROJCTRA	*PGM	CORPPGM	RPGLE	Project List
PROJCTDA	*FILE	CORPPGM	DSPF	Project List
PROJCTRB	*PGM	CORPPGM	RPGLE	Project Form
PROJCTDB	*FILE	CORPPGM	DSPF	Project Form

Object	Type	Library	Attribute	Description
List programs				
ACTPRJRA	*PGM	CORPPGM	RPGLE	List - Activities
ACTPRJDA	*FILE	CORPPGM	DSPF	List - Activities
DEPLISRA	*PGM	CORPPGM	RPGLE	List - Departments
DEPLISDA	*FILE	CORPPGM	DSPF	List - Departments
EMPLISRA	*PGM	CORPPGM	RPGLE	List - Employees
EMPLISDA	*FILE	CORPPGM	DSPF	List - Employees
EMPPRJRA	*PGM	CORPPGM	RPGLE	List - Employee Activities
EMPPRJDA	*FILE	CORPPGM	DSPF	List - Employee Activities
PRJLISRA	*PGM	CORPPGM	RPGLE	List - Projects
PRJLISDA	*FILE	CORPPGM	DSPF	List - Projects

PC Organizer

EDITDOCC	*PGM	CORPPGM	CLLE	Edit document
SHOWDOCC	*PGM	CORPPGM	CLLE	Show document

OS Command Call

OS4DSPMSG	*PGM	CORPPGM	CLLE	OS/400 - DSPMSG
OS4QCMD	*PGM	CORPPGM	CLLE	OS/400 - CALL QCMD
OS4WRKSPLF	*PGM	CORPPGM	CLLE	OS/400 - WRKSPLF

Menu System

CORPMNU	*MSGF	CORPPGM		
CORPMNU	*FILE	CORPPGM	DSPF	
CORPMNU	*MENU	CORPPGM	DSPF	
SETLIBLC	*PGM	CORPPGM	CLLE	Set library list

Source Files

Object	Type	Library	Attribute	Description
QCLSRC	*FILE	CORPPGM	PF	
QDDSSRC	*FILE	CORPPGM	PF	
QMNUSRC	*FILE	CORPPGM	PF	
QRPGLESRC	*FILE	CORPPGM	PF	

The demo programs showcase a wide variety of functionalities supported by IceCap. Key variations include:

Toggle Colors	Test various color themes, including normal and reversed text.
Tags	Codes placed in parentheses immediately after related fields: • (DMYY) - Calendar field • (Y/N) - Checkbox • (F, M) - Radio buttons explaining the abbreviations • (12-20) - ComboBox with predefined values • (JOB) - ComboBox retrieving values via an SQL statement • (MAPS) - Map search based on field content • (W) - Internet search on field content • (>) - Calls a 5250 program • F4 - Prompts a field with an icon.
F-Keys	Calls various programs and commands, including: • F18 - Opens Notepad for document editing. • F19 - Opens a PDF viewer for document reading.
Naming	Variation in F-key naming, such as "F3-Exit" vs. "Cmd 3 - Exit."
Column Headings	Differ between one or two-line headings.
Forms	Various opening formats, including windowed or fullscreen.
Full-Screen Modes	Using both 24x80 and 27x132 screen sizes.

Configurations and Components

The Demo Environment also relies on specific configurations and components that are critical to its functionality.

Configurations	Located in the directory: /www/iceapp/icecap2/
Icecap_config.xml	The Demo Environment is referenced under the following configurations: corplex , corpflow , and corpflip . These configurations are defined in the Icecap_config.xml file, and the emulation and navigation components listed below are explicitly mentioned in this configuration section.
Components	Located in the directory: /www/iceapp/icecap2/plugins/
emlFlxCorp.rpgle	The Emulation component for Flex mode in the CORPDATA Demo Environment. It provides two extra tabs on the Employee Form.
emlFlwCorp.rpgle	The Emulation component for Flow/Flip mode in the CORPDATA Demo Environment. It provides three extra tabs on the Employee Form. For more technical details, refer to the chapter "Integration."
navCapCorp.rpgle	The Navigation component, used to jump directly to a specific Employee in either Display or Edit mode.

These configurations and components ensure the enhanced functionality and flexibility of the Demo Environment, allowing users to experience and interact with the advanced features of IceCap.

Portfolio

Portfolio is an essential component of IceCap that greatly enhances the usability and functionality of your ERP solution. It provides a modern, browser-based interface that transforms traditional 5250 applications into a more intuitive and efficient environment.

What is Portfolio?

Portfolio is a comprehensive solution designed to modernize the interface of IBM i applications. It enables users to interact with their ERP systems through a web browser, delivering a unified and user-friendly experience. Portfolio supports both a browser-based interface and a client version that can run on users' computers, offering flexibility and accessibility.

Key Features of Portfolio

- **Web Enablement**
Builds on the transformation achieved with IceCap, incorporating new web applications into a modern web solution, thereby overcoming the limitations of traditional emulation.
- **Unified Modern Solution**
Seamlessly integrates IceCap web applications into a cohesive, modern interface, making navigation and task execution easier for users.
- **Tree Structure Organization**
Organizes applications into systems and further categorizes them into tree structures for easy access and efficient navigation.
- **Modern Multi-tasking**
Supports multitasking with tabs and windows, allowing users to switch between tasks effortlessly.
- **Enhanced User Management**
Includes comprehensive user management, with features for password administration and governance of roles and permissions for both IBM i and other users.
- **Improved Workflow**
Streamlines workflows by merging genuine web applications with IceCap web applications into a single, unified interface.
- **Integration Capabilities**
Enables the embedding of transformed 5250 programs into other web applications,

integrates them with cloud services, and connects them with various systems or platforms.

Setting up Portfolio

To implement Portfolio after installation, follow these steps:

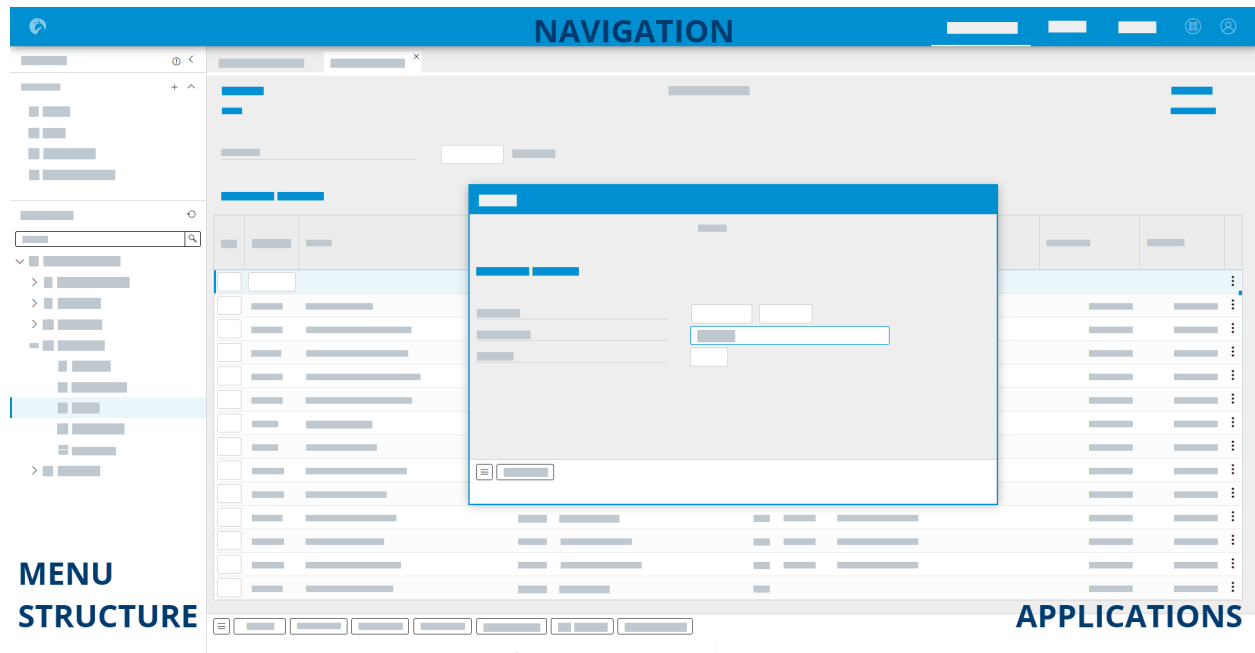
1. **Configuration** - Configure Portfolio by setting up the menu structure, user roles, permissions, and other administrative settings. This setup can be done through the SYSTEM menu in Portfolio. Refer to the “System” chapter to get more details.
2. **Integration** - Integrate Portfolio with existing ERP systems and other applications to provide a seamless user experience. Refer to the “Menu-Loader” section to get more details.
3. **Customization** - Customize the visual appearance of the interface using the available configuration options. This includes setting up the color themes, company logo, login screen, and system naming. Refer to the “Customizing Portfolio” section to get more details.

Using Portfolio

The workspace of Portfolio features a structured layout divided into four main panels:

1. **Navigation Panel** (Top) - Provides access to system selection, desktop, tab and window management, and user settings, allowing users to efficiently navigate and organize their workspace.
2. **Menu Structure Panel** (Left) - Organizes applications into a hierarchical tree structure, enabling easy access and quick launching of applications through a customizable menu system.
3. **Application Panel** (Middle) - The central area where applications run in tabs or windows, supporting multitasking and providing an intuitive way to manage and interact with multiple applications simultaneously.
4. **Help Panel** (Right) - Displays context-sensitive help related to the active application, aiding users in finding information and guidance quickly.

This workspace is designed to enhance productivity by providing a cohesive, modern interface that integrates traditional 5250 applications into a more intuitive and flexible environment.



Using the Navigation Panel

The Navigation Panel in Portfolio is designed to help users efficiently manage and navigate through their applications and systems. Below are the key aspects of the Navigation Panel:

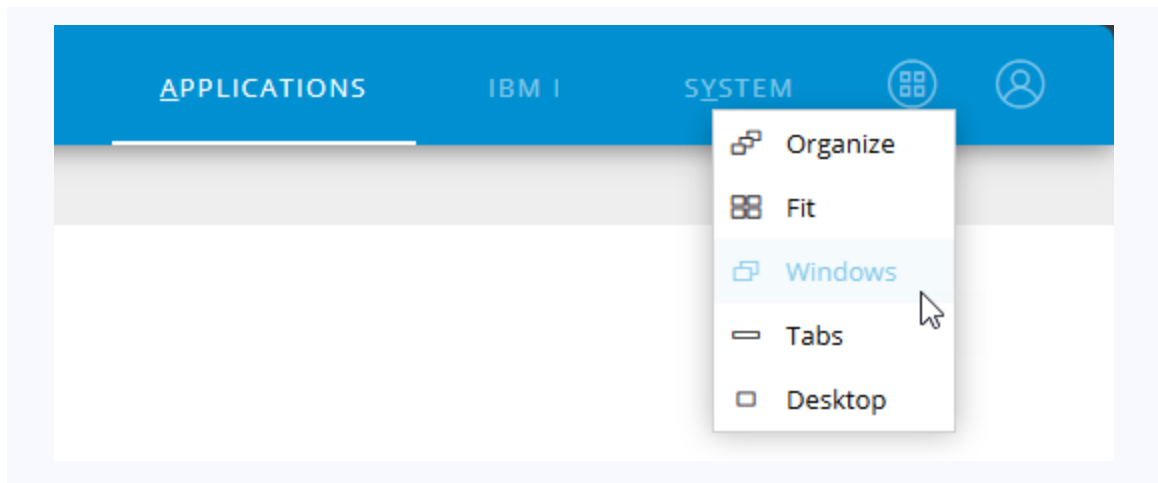
Overview of the Navigation Panel

- **Location** - The Navigation Panel is positioned at the top of the Portfolio main window, providing a central control point for navigating through the various systems and applications within the Portfolio environment.
- **Purpose** - It allows users to select and arrange systems, tabs, windows, and user settings, facilitating quick access to the required functions and improving overall workflow efficiency.

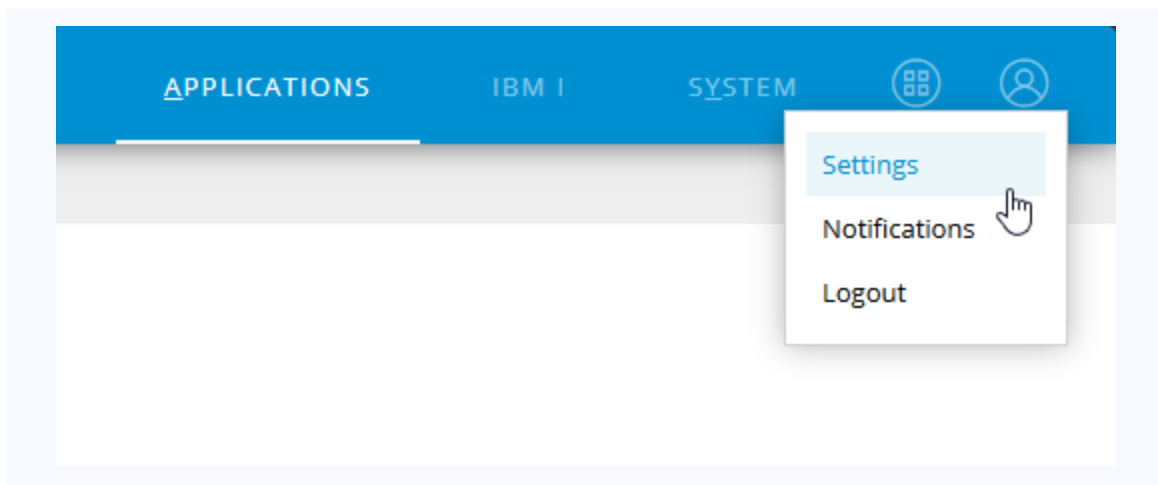
Features of the Navigation Panel

- **System Selection** - Users can choose different systems or subsystems, enabling them to quickly access the specific applications they need.
- **Tab and Window Management** - The Navigation Panel facilitates the management of tabs and windows, allowing users to seamlessly switch between open applications, arrange them side by side, or organize them to suit their workflow. Users can even temporarily save their

current setup as a "Desktop," enabling them to quickly switch between different Desktops and start new workflows without clearing the Application Panel.



- **User Settings** - The panel provides access to user settings, where users can adjust preferences, manage their profile, and customize the interface according to their needs.



Practical Use

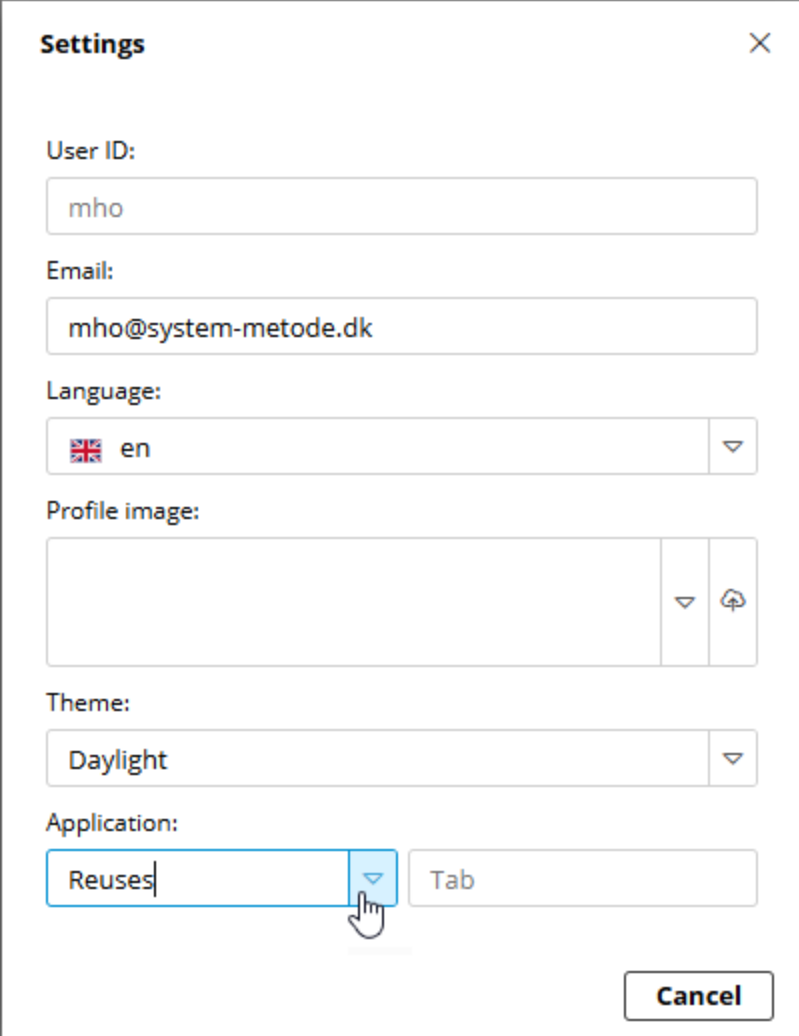
- **Enhanced Workflow** - The Navigation Panel streamlines the user experience by providing easy access to all necessary tools and applications, reducing the time needed to switch between tasks.

- **Customization** - Users have the flexibility to arrange their workspace within the Navigation Panel, enhancing productivity and ensuring that frequently used applications are readily accessible.

The Navigation Panel is designed to improve the usability of the Portfolio environment, making it easier for users to interact with multiple systems and applications in a cohesive and efficient manner.

Launching Applications

By default, Portfolio attempts to "reuse" already open tabs or windows when launching applications. Users have the option to change this behavior, allowing a new tab or window to open each time an application is launched. While this promotes multitasking, having multiple instances of the same application open can lead to a cluttered workspace, making it more difficult to keep a clear overview.



The image shows a 'Settings' dialog box with a close button (X) in the top right corner. It contains several input fields and dropdown menus. The 'User ID' field contains 'mho'. The 'Email' field contains 'mho@system-metode.dk'. The 'Language' dropdown shows a flag icon and 'en'. The 'Profile image' field is empty with a dropdown arrow and a refresh icon. The 'Theme' dropdown shows 'Daylight'. The 'Application' dropdown shows 'Reuses' with a hand cursor pointing to the dropdown arrow. To the right of the 'Application' dropdown is a 'Tab' field. A 'Cancel' button is at the bottom right.

Settings ✕

User ID:

Email:

Language:
 ▼

Profile image:
 ▼ ↻

Theme:
 ▼

Application:
 ▼

Cancel

Using the Menu Structure Panel

The Menu Structure Panel is an integral part of the Portfolio environment, providing a structured and organized way for users to access applications and navigate through the system. Below are the key details about the Menu Structure Panel:

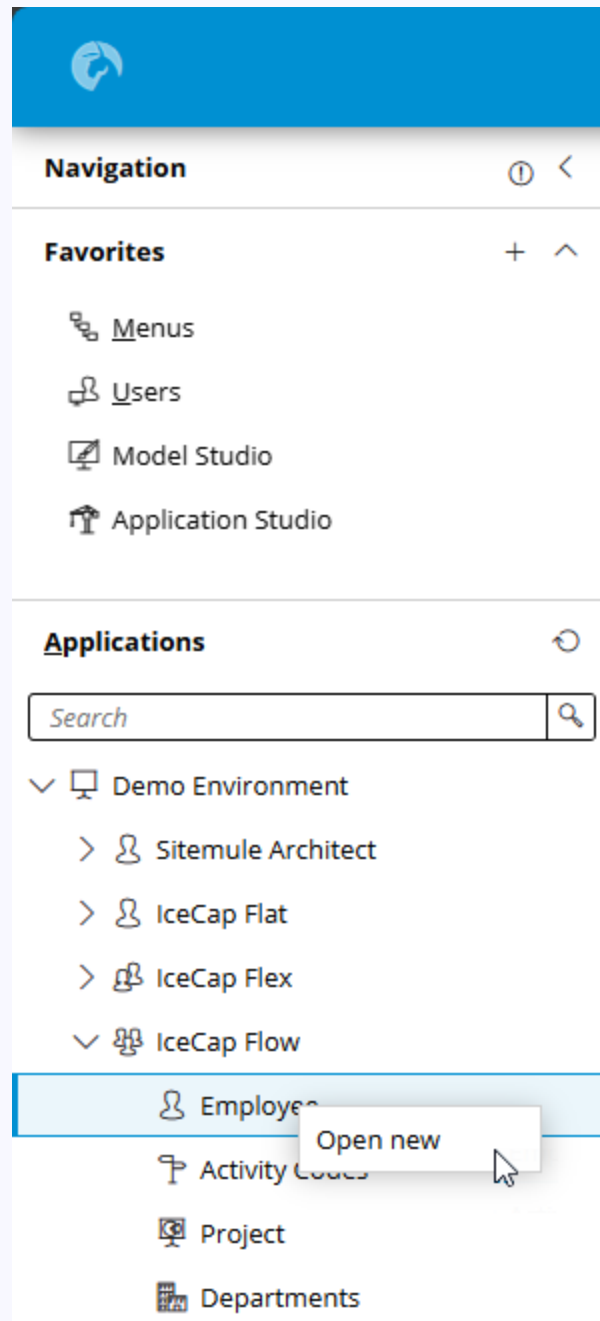
Overview of the Menu Structure Panel

- **Location** - The Menu Structure Panel is located on the left side of the main Portfolio window. It can be expanded or collapsed as needed, providing flexibility in how much of the workspace it occupies.

- **Purpose** - This panel serves as the primary navigation hub within the system, where users can find and launch applications. It organizes the available applications into a tree structure, making it easy to locate specific tools or functions.

Features of the Menu Structure Panel

- **Tree Structure Organization** - Applications are organized into a hierarchical tree structure. This organization reflects the structure of systems and subsystems, enabling users to drill down through menus to find the desired application. The Menu Structure Panel can be hidden by clicking the "<" at the top, providing more space for the Application Panel.
- **Search feature** - Each section of the menu includes an optional search feature, allowing users to quickly find applications by typing keywords. This is particularly useful in large systems with many applications.
- **Favorites section** - Users can drag frequently used applications into a separate "Favorites" section. This feature allows quick access to commonly used tools, improving efficiency.
- **Click and Right-click functionality**
 - Clicking on an application opens it in a new tab or switches to an already open tab with that application.
 - Right-clicking on an application allows it to be opened multiple times in new tabs, enabling users to run multiple sessions of the same application side by side



This example illustrates how to open an application that already is opened in Application Panel

Practical Use

- **Workspace Management** - The Menu Structure Panel helps users manage their workspace effectively by providing easy access to all necessary applications. Users can customize the panel by organizing their favorite applications and using the search feature to quickly find what they need.
- **Flexibility** - The panel can be hidden or collapsed when more screen space is needed for the Application Panel, offering flexibility depending on the user's current task.

Customization

- The structure of the menu and the available applications can be customized by administrators through the system settings. This ensures that the Menu Structure Panel aligns with the specific needs of the organization and its users.

Favorites

- **Drag-and-Drop** - The Favorites menu is located at the top of the Menu Structure Panel, offering a convenient way to access frequently used applications. To add an application to your Favorites, drag and drop it into the Favorites section.
- **Customization** - Favorites can be reorganized through drag-and-drop and further organized by creating folders based on your workflow. Folders can be configured to Run at Startup, allowing essential applications to launch automatically when you log in to Portfolio. This feature streamlines your workflow by ensuring key applications are ready without navigating the menu.
- **Cross-System Consistency** - Favorites are personalized and saved with your user profile. The Favorites menu remains consistent across all systems within the Top Navigation Panel, optimizing your work by reducing the need to switch between systems.

The Menu Structure Panel is designed to enhance the user experience by providing an intuitive and efficient way to navigate through the Portfolio environment, helping users to access the tools they need with minimal effort.

Using the Application Panel

The Application Panel in Portfolio is designed to manage and display applications in a modern, user-friendly interface. Below are the key aspects of the Application Panel:

Overview of the Application Panel

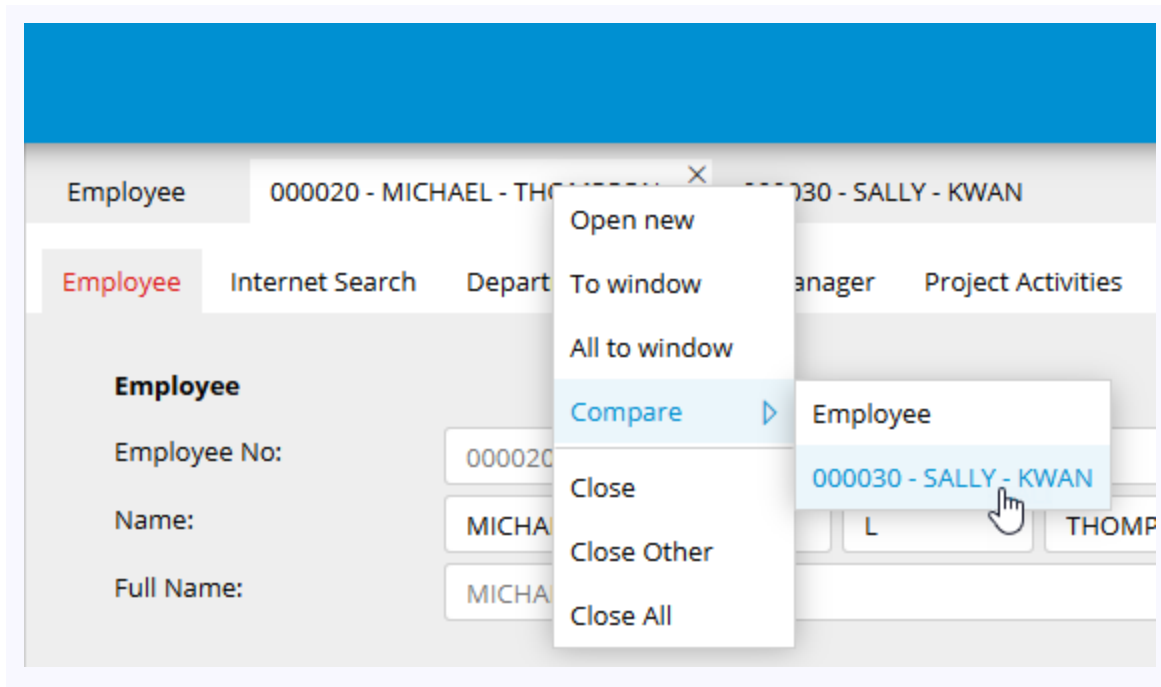
- **Purpose** - The Application Panel is where all applications are displayed and managed within the Portfolio environment. It operates within its own tab or window, allowing users to interact with various applications simultaneously.
- **Tab Management** - Each application opens in a separate tab or window within the Application Panel. Users can easily switch between tabs, close them, or manage multiple applications concurrently. Right-clicking on a tab provides additional options for tab management.
- **Window Management** - The Application Panel supports windowed applications, enabling users to compare, resize, and arrange windows according to their needs. This multitasking capability is essential for users working with multiple applications or datasets.

Features of the Application Panel

- **Right-click functionality** - Right-clicking on an application allows it to be opened multiple times in new tabs, enabling side-by-side comparisons of different 5250 sessions or other applications.
- **Scroll through tabs** - If many tabs are open, small arrows on either side of the tabs allow users to scroll through them easily.
- **Closing Applications** - Tabs or windows can be closed by clicking the "X" in the upper right corner, helping users manage their workspace efficiently.

Practical Use

- **Multitasking Environment** - The Application Panel transforms the traditional ERP interface into a multitasking environment, enabling users to handle multiple tasks at once without the limitations of the traditional 5250 interface.
- **Flexibility and Control** - Users have full control over their workspace, with the ability to customize how applications are viewed and managed within the panel, whether in tabs or separate windows.



How to compare tabs and windows

To effectively compare windows within the Portfolio environment, you can utilize the system's multitasking and window management features. The following steps outline how to efficiently compare different windows or tabs:

1. **Open multiple tabs or windows** - Begin by opening multiple applications or screens in different windows or tabs using the standard navigation options. You can right-click on an application in the menu to open it in a new tab or window, allowing you to work with multiple instances simultaneously.
2. **Switch between windows** - Use the keyboard shortcut **ALT + Z** to quickly cycle through your open windows. This enables you to effortlessly switch between the windows you wish to compare.
3. **Arrange windows side by side** - Manually drag windows to position them side by side on your screen or use the "Fit" in Tab, and Window Management in the Top Panel. This arrangement facilitates easy comparison of content across different windows.
4. **Use tabs for comparison** - If you need to compare different parts of the same application or screen, you can open these in separate tabs. Use the shortcut **ALT + 0-9** to switch between tabs or **ALT + 1/2** to cycle through them rapidly.

5. **Compare with a specific tab** - Right-click on the active tab, select "Compare," and choose the tab you want to compare it with. The selected tabs will automatically switch from tabs to windows and be arranged side by side for immediate comparison.
6. **Convert tabs to windows and vice versa:** You can convert a tab into a standalone window using **ALT + W** or bring a window back into the tabbed interface with **ALT + T**. This flexibility allows you to organize your workspace according to the type of comparison you need to make.

Most of these shortcuts can also be accessed by right-clicking on tabs, using the window control buttons, or the arranging icon in the top-right corner of the Portfolio interface. Refer to the appendix covering Shortcuts.

Example

For instance, if you need to compare customer records across two different screens:

- Open the first record in one tab or window.
- Open the second record in another tab or window.
- Arrange these windows side by side or use the relevant shortcuts to switch between them.

This approach allows you to visually compare data, making it easier to identify differences or similarities between the records. These tools and shortcuts significantly enhance your ability to manage and compare multiple windows and tabs efficiently within the Portfolio environment.

Using the Help Panel

The Help Panel in Portfolio is a feature designed to provide users with context-sensitive assistance and support directly within the Portfolio environment. Here's an overview of its key aspects:

Overview of the Help Panel

- **Location** - The Help Panel is located on the right side of the main Portfolio window. It can be accessed whenever users need guidance or additional information related to the application they are currently using.

- **Purpose** - The primary purpose of the Help Panel is to offer real-time help and documentation that is relevant to the active application, enhancing user productivity and reducing the need for external support resources.

Features of the Help Panel

- **Context-Sensitive Help** - The Help Panel automatically displays help content that is relevant to the application or task currently active in the Application Panel. This means users receive immediate support without having to search through external manuals or documentation.
- **Customizable Content** - Administrators or users with the appropriate permissions can create, edit, and customize the help content displayed in the Help Panel. This allows organizations to tailor the help resources to their specific needs and ensure that the information provided is accurate and useful.
- **On-Demand Access** - The Help Panel can be opened or closed as needed, giving users control over when they access the information. This helps to keep the workspace uncluttered while still providing easy access to support when required.

Practical Use

- **Improved User Experience** - By offering instant access to relevant help information, the Help Panel reduces the learning curve for new users and supports more efficient problem-solving for all users.
- **Enhanced Productivity** - With help content readily available within the workspace, users can resolve issues quickly without leaving the application, minimizing disruptions to their workflow.

Deactivation Option

- **Default Setting** - The Help Panel is deactivated by default but can be activated by users or administrators depending on the organization's needs.

The Help Panel is an integral part of the Portfolio environment, designed to enhance user support and improve overall efficiency by providing immediate, relevant assistance within the workspace.

Linking to Applications in Portfolio

The linking feature in Portfolio allows users to generate a direct link to a specific application and share it with another user. When the recipient clicks on the link, they will either be taken directly to the application if already logged in, or guided through the login process first. This provides a fast and efficient way to access a specific part of the system.

For applications developed with Model Studio, the link can include selected data that the recipient should focus on. This makes it even more effective by allowing the recipient to arrive at the exact data point, enhancing usability and collaboration.

Use Case

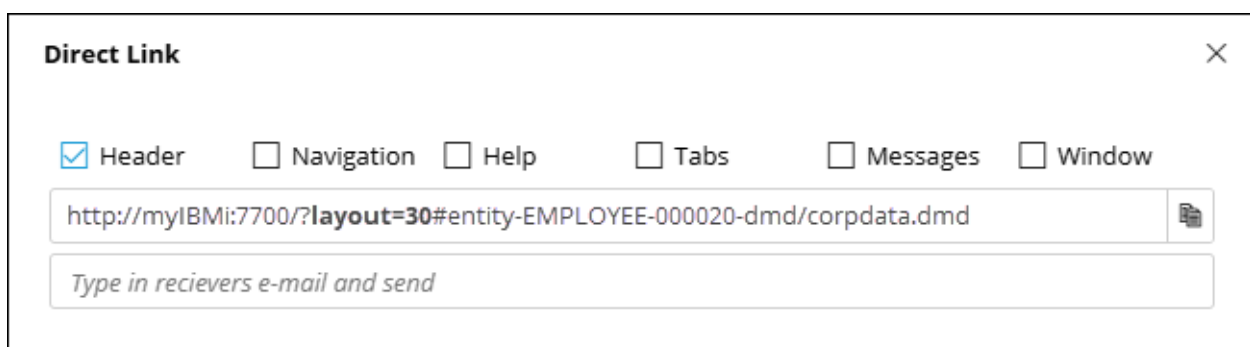
Imagine a situation where a user needs help filling out information for a new employee. By creating a link to the employee form, the employee number can be included in the link. When the recipient clicks the link, they will be taken not just to the application, but directly to the form with the relevant employee data already loaded.

Creating a Link

To create a link, simply click on the Settings Menu located in the upper right corner of the form and select Link. This will generate a link to the current application, including any relevant data such as the employee ID you are working with.

Customizing the Layout

When creating the link, you can define which components of the interface the recipient will see. For example, the recipient might only need access to the Navigation Top Panel and the Application Panel where the Employee ID 000020 is shown. In this case, setting `layout=30` ensures that the recipient sees only these components, while the navigation structure and other elements are hidden. By streamlining the interface and focusing on key elements, this approach helps users efficiently complete their tasks and close the tab once finished.



The screenshot shows a 'Direct Link' dialog box with a close button (X) in the top right corner. It contains several checkboxes for interface components: 'Header' (checked), 'Navigation' (unchecked), 'Help' (unchecked), 'Tabs' (unchecked), 'Messages' (unchecked), and 'Window' (unchecked). Below these is a text input field containing the URL: `http://myIBMi:7700/?layout=30#entity-EMPLOYEE-000020-dmd/corpdata.dmd`. At the bottom is another text input field with the placeholder text 'Type in recievers e-mail and send'.

System

The System Settings in Portfolio is designed to streamline the management and organization of your IBM i applications. It allows administrators to define and manage user roles, permissions, and access controls, ensuring that users have the appropriate level of access to perform their tasks efficiently.

Within the System module, you can create and customize menu structures, organize applications into logical groupings, and configure various system settings to optimize the user experience. This feature also supports the integration of traditional 5250 menu systems into a modern, web-based interface, providing a seamless transition for users.

The System feature enhances the overall functionality and usability of Portfolio, making it easier to maintain and navigate your application environment.

Considerations and recommendations

Screen Resolution and Microsoft Windows Setup

Microsoft recommends setting your display resolution to the native resolution of your monitor, often labeled as “Recommended” in the display settings. The native resolution is specifically optimized to provide the best clarity and sharpness for text, images, and other screen elements.

In addition to adjusting the resolution, Microsoft advises modifying the scaling settings to ensure that text and other items are appropriately sized on your screen. This is particularly important when using high-DPI displays, such as 4K monitors, where text and icons may appear too small at the native resolution. Scaling options can be found under “Scale and layout” in the Windows display settings, where you can increase the size of text, apps, and other items by selecting a higher percentage, such as 125% or 150%.

When using multiple monitors, it is essential to maintain consistent scaling settings across all displays to avoid issues where text or UI elements appear blurry or improperly sized when moving between screens. Consistent scaling ensures a seamless experience across different monitors.

For the best and sharpest presentation of IceCap and Portfolio, it is crucial to follow these recommendations. IceCap is optimized for a resolution of 1920x1080 pixels, displaying 27 lines with 132 characters per line in Portfolio. If your Windows settings use a lower resolution or increased text

size (e.g., 125%), you can compensate by adjusting the zoom level in your browser.

Using the Zoom Function

To adjust the zoom level in the standard browser, you can follow these steps:

Zoom In/Out Using Keyboard Shortcuts:

- To zoom in (enlarge content), press **Ctrl +** (plus key).
- To zoom out (shrink content), press **Ctrl -** (minus key).
- To reset the zoom level to 100%, press **Ctrl 0** (zero key).

Zoom In/Out Using the Mouse:

- Hold down the **Ctrl** key on your keyboard and scroll the mouse wheel up to zoom in or down to zoom out.

Zoom In/Out from the Menu:

1. Click on Settings in the upper right corner of the browser to open the menu.
2. In the “Zoom” section, use the **+** and **-** buttons to adjust the zoom level.

Set a Default Zoom Level:

1. Open the browser menu by clicking on the three dots in the upper right corner.
2. Select Settings.
3. Scroll down and click on Appearance.
4. Under the “Zoom” section, you can set the default zoom level for all pages.

As an alternative to adjusting the zoom level, we recommend installing the Portfolio Client for an optimized experience.

Web Applications as a Client

The Portfolio Client is an application that is installed natively on computer whether it's Windows, Linux, or macOS. One of its key features is the ability to automatically adjust the zoom level of the

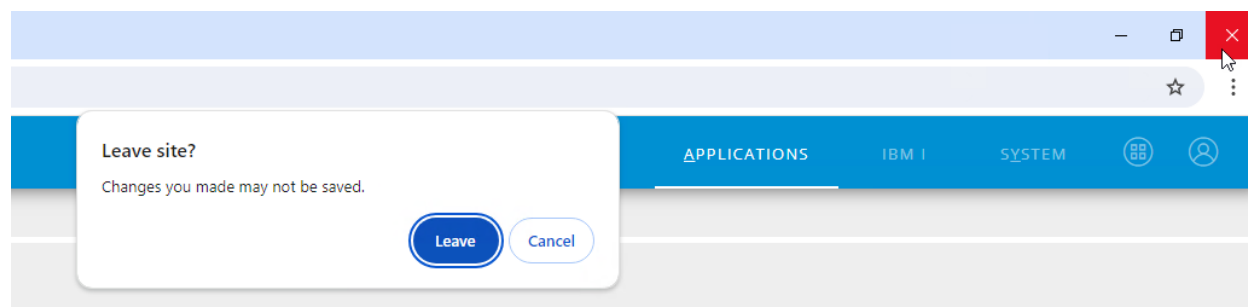
client content to the optimal size, even recalibrating the zoom percentage when the window is moved to another screen with a different resolution.

For more detailed information, please refer to the chapter "Portfolio Client Manager."

Loss of Connection and Forced Termination of a Web Application

This chapter explains the behavior of the application when the connection is lost or when the browser, a browser tab, or Portfolio tabs or windows are forcefully closed using the "X" in the upper right corner.

Portfolio includes a built-in safeguard that activates immediately if a user attempts to leave while one or more applications are running. A prompt, "Leave site? Changes you made may not be saved," is displayed, urging the user to assess whether any unsaved data could be lost before proceeding.



Native Web Applications

Native Web Applications are designed to handle unexpected loss of connection or forced browser termination. While data loss might occur, the application is generally equipped to manage such situations and continue operation without causing further concern to the user.

5250 IceCap Applications

Handling a 5250 program is more complex when a user abruptly exits. This situation is analogous to terminating an emulation program using the "X" button rather than the appropriate function keys to exit.

Closing Portfolio tabs and windows

When a tab or window running a specific application is closed, the application is terminated, and any unsaved work will be lost. Additionally, closing a tab or window logs the user out of the 5250 program. This can result in an improper exit, potentially leaving the 5250 program in a state that may require administrative intervention. To prevent this, you can remove the "X" button from tabs and windows for specific or all 5250 programs. This is achieved by setting the property `closable:false` when adding the program to the menu.

Closing Browser tabs or the entire Browser

The scenario remains the same; however, Portfolio cannot remove the "X" button in the browser itself and must rely on the safeguard message to alert the user.

Abrupt Termination of 5250-based Web Applications

Depending on the circumstances, Portfolio may either terminate the 5250 program immediately when the user exits or later, when the connection between the active 5250 program and Web Applications cannot be verified. Portfolio will automatically terminate any 5250 programs that have lost connection after 60 seconds.

Menu Setup

The "Menu Setup" chapter provides detailed instructions on how to configure and manage the menu structure within the Portfolio environment. A well-organized menu system is essential for enhancing user navigation and ensuring that applications and functions are easily accessible. This chapter will guide you through the process of setting up, editing, and customizing the menu structure, including the addition of groups, links, and applications, as well as defining roles and access levels.

Accessing the Menu Setup

Menu configuration is performed within the system settings of Portfolio. When you enter the menu setup interface, you will see the entire menu structure displayed hierarchically. This hierarchy consists of various levels, each serving a specific organizational purpose within the menu.

Hierarchical Structure

The menu structure in Portfolio is organized into a multi-level hierarchy comprising several types of levels:

1. **Group:** Acts as a folder within the menu structure, organizing related links and applications. Groups can contain other groups, links, and applications.
2. **Link:** A navigational element that directs users to a specific URL or function within the system.
3. **Application:** Represents the actual programs or tools that users interact with. There are several types of applications, including:
 - **Model Studio Applications:** Generated through the Model Studio.
 - **Application Studio Applications:** Created within the Application Studio.
 - **IceCap Applications:** 5250 programs transformed into web applications by IceCap.
 - **Other Applications:** Any additional applications within the hierarchy.

System Level

The top level of the hierarchy is always a Group that represents a System. This level is displayed in the top Navigation Panel, and its contents are shown in the Menu Structure Panel when accessed. Typically, 4 to 6 groups are present at the system level, each representing different systems within

the organization. Additionally, this top level includes a user-friendly feature that allows for the searchable navigation of the underlying tree structure.

Adding Menu Items

To add a new item to the menu structure, right-click on the desired level and select "Add." You can add a group, link, or application at any level, except the top (system) level. Additionally, you can edit, delete, or copy existing items as needed.

- **Adding a Group:**
 - **Content Tab:**
 - **Title:** Enter the title of the group. You can make this title language-specific by clicking the globe icon, which allows you to input the title in different languages.
 - **Icon:** Use the search tool to select an appropriate icon for the group.
 - **System:** (Optional) By setting a system environment for a group, all applications within the group will adopt the environment settings, such as language and other custom configurations.
 - **URL:** (Optional) This can be used if the group should open a specific feature, such as a search function or a menu-builder.
 - **Start Expanded:** Choose whether this group should be expanded or collapsed by default when the menu loads.
 - **Shortcuts:** Assign a keyboard shortcut to quickly access this group. Refer to "Customized Shortcuts" in Appendix C, "Keyboard Shortcuts," for more details.
 - **Roles Tab:**
 - Define which user roles have access to this group. Only users with the specified roles will be able to view and interact with the group.
- **Adding a Link:**
 - Adding a link is similar to adding a group but serves the purpose of directing users to a specific URL or system feature.
- **Adding an Application:**
 - When adding an application, the form provided will depend on the type of application you are registering. Applications from Model Studio, Application Studio, and IceCap are added through a form divided into three sections: Menu Location,

Application, and Roles. Other generic applications use a form similar to that for adding Groups and Links.

- Example of Adding an IceCap Application:
 - **Menu:** Displays the location in the menu to confirm where the "add" operation is taking place.
 - **Application:** Provides details specific to the type of application (e.g., IceCap):
 - **Title:** As with groups, the title can be language-specific.
 - **Icon:** Search and select an icon for the application.
 - **Initialization (optional):** Specify any initialization command or program that should run before the 5250 program is called.
 - **Function:** Enter the 5250 program or command to be executed.
 - **Config:** Choose a presentation mode (e.g., Flat, Flex, Flow, Flip) or a custom configuration.
 - **Roles:** Assign access rights to specific user roles.

Editing Menu Items

To modify an existing menu item, right-click on it and select "Edit." You have two editing options:

- **Standard Edit:** Modify basic settings such as the title, icon, and access roles - similar to the process described above.
- **Advanced Edit:** Offers additional configuration options, such as adjusting how the application runs or adding parameters to its configuration. For example, you can specify that an application cannot be closed using the tab/window's close button by setting the `closable:false` parameter, requiring the use of specific commands like F3 or F12 to exit.

Drag-and-Drop Functionality

Portfolio supports drag-and-drop functionality for rearranging the menu structure. You can move groups, links, or applications within the hierarchy, including entire folders with multiple applications. This feature allows for easy reorganization of the menu to meet changing needs.

IBM i Command Menu

Portfolio includes a predefined menu that mirrors the structure of the command menus used in the IBM i operating system. This menu functions similarly to a customized Menu-Loader and demonstrates how an ERP solution's menu system can be effectively presented and structured. One of the key enhancements is an intelligent search function designed to improve workflow efficiency.

In addition to searching by command description, users can enter a command or part of a command to locate and prompt the command. For example, typing "DSP" will retrieve all commands containing **DSP**, such as **DSPMSG** and **CRTDSPF**. If a full command is entered, the command prompt will appear immediately. Users can also input parameters to be prompted along with the command. This feature integrates the search field with the popular Command Entry on IBM i.

All command operations adhere to the authorization system of IBM i, ensuring that users can only prompt commands for which they have access when operating traditional 5250 screens.

This approach can also be applied to a customized Menu-Loader for launching programs identified by unique identifiers, such as **ORDER** or **O100** for order entry. Similar to commands, the Menu-Loader should verify user authorization before starting an application.

Setup Menu-Loader

By dynamically generating easy-to-navigate menus, the Menu-Loader enables users to interact with their familiar ERP systems through a modern browser interface. It offers customization options that allow the inclusion of specific programs and commands tailored to meet organizational needs. The Menu-Loader simplifies the transition to a web-based system, improves usability, and increases workflow efficiency.

Customizing the Menu-Loader

Portfolio comes with an installed template for creating a customized Menu-Loader program. This template demonstrates how straightforward it is to develop your own Menu-Loader. The template program is written in RPG Free and utilizes SQL to read from the file `<PortfolioLibrary>/MENULoader` on IBM i.

The "MENULoader" system is displayed as a separate section in the Navigation Panel. If it is not visible, you need to grant Administrator access by adding the appropriate role to the menu structure in the System Settings.

When you click on "MENULoader," the menu structure is dynamically built using the template program, displaying two sections with three applications each. The structure is defined by the MENU_ID and PARENT_ID, as illustrated in the example below:

MENU_ID	PARENT_ID	TITLE	LINK	ICECAP_INITFUNC	ICECAP_MODE
1		Links on the internet			
2	1	Bing	https://www.bing.com		
3	1	Wikipedia	https://wikipedia.org		
4	1	Maps	https://www.openstreetmap.org/node/13707878		
5		IBMi Commands			
6	5	Work Spooled File	WRKSPLF SELECT(*ALL)		flow
7	5	Work System Status	WRKSYSSTS		
8	5	Command Entry	CALL QCMD		

Integrating the Menu-Loader

There are two approaches to integrating the Menu-Loader with your ERP menu system on IBM i:

1. **The Easy Way** - Convert your existing menu files into the MENULoader format:
 - **MENU_ID** - Integer, the unique identifier for each menu item.
 - **PARENT_ID** - Integer, indicates the parent menu item for creating nested structures.
 - **TITLE** - Varchar(256), the title of the menu item.
 - **LINK** - Varchar(8192), the link or command associated with the menu item.
 - **ICECAPINITFUNC** - Varchar(256), specifies the initial function in IceCap.
 - **ICECAPMODE** - Varchar(100), determines the mode in which IceCap operates.

2. **The Correct Way** - Modify the template program to directly use your existing menu files.
This method involves customizing the RPG Free program to fit your specific menu structure requirements.

For more detailed technical information and guidance on modifying the template program, refer to Appendix F, "Customized Menu-Loader."

Using the Menu-Loader

To configure and use the Menu-Loader, follow these steps:

1. **Open Menu Setup** - Navigate to the "Menu Setup" section within the System Settings.
2. **Add a Group** - Select the desired level within the menu structure where you want to add the Menu-Loader, and create a new group.
3. **Register the Menu-Loader** - Enter `/menuloader.aspx` in the URL field.
4. **Save and Refresh** - Save the newly created group, and then refresh the menu structure to apply the changes.

These steps will successfully integrate the Menu-Loader into your menu structure.

This flexible and powerful tool allows you to maintain a seamless user experience while transitioning to a modern, web-based interface, ensuring that your IBM i applications remain accessible and easy to navigate.

User Management

User Management is a fundamental aspect of maintaining an organized and secure environment within the IceCap and Portfolio system. This chapter provides detailed instructions on how to manage user accounts, including the creation, modification, and deactivation of users. It also covers how to assign roles and permissions to users, ensuring that each individual has the appropriate access levels and capabilities necessary for their role.

Effective user management is essential for ensuring that the system operates efficiently and securely. By carefully managing user access and responsibilities, administrators can prevent unauthorized access, maintain compliance with security policies, and streamline operations.

In this chapter, you will learn how to set up new users, manage existing user profiles, assign roles and permissions, and troubleshoot common issues related to user management. Whether you are onboarding new team members or updating access rights for current users, the guidelines provided here will help you maintain control over your system's user base.

User Management Grid

To begin managing users, navigate to the system menu and select the Users application. Upon entering the application, a complete list of all users currently registered in the system will be displayed.

Searching for Users

If there is a large number of users, utilize the search bar located in the upper left corner to efficiently locate a specific user.

Creating a New User

To create a new user, click the Create button. This action opens a form where you can input the necessary details for the new user account.

Actions and Options

Right-clicking on a user within the list reveals a context menu with several actions:

- **Edit** - Opens the user's profile for editing.
 - **Delete** - Removes the user from the system.
 - **Copy** - Duplicates the user profile.
 - **Change Password** - Opens a small window where the new password can be entered and confirmed by retyping.
 - **Status** - Toggle the user's status between enabled, disabled, or locked.
 - **Mail** - Dispatch an email prompting the user to reset their password.
-

User Management Form

Double-clicking on a user or selecting "Edit" from the context menu will open the user's profile for editing. The user profile is organized into three tabs:

- **Content** - Displays and allows editing of the user's personal information and account details.
- **Roles** - Manages the roles assigned to the user.
- **History** - Logs all changes made to the user's account.

Content Tab

The Content tab includes several fields where detailed user information is managed:

- **User ID** - The unique identifier used for system login.
- **Name** - The full name of the user.
- **Password** - The password is encrypted and must be created by the administrator through the right-click menu in the Grid or by sending an invite email to the user. This field is protected.
- **Image** - An image that can replace the default icon in the upper right corner of the Navigation Panel, displaying an avatar or photo of the user.
- **Email** - Important for account recovery and communication.
- **User Profile** - Used specifically for users who do not have an IBM i user profile, typically referencing a group user profile on IBM i with the necessary authorities for accessing applications.
- **Last Logged On** - Shows the date and time of the user's last login.
- **Status** - The current status of the user account.
- **Language** - The preferred language for the user interface.

Roles Tab

In the Roles tab, the user's assigned roles are displayed. The system includes three predefined roles:

- **Administrator**
- **Superuser**
- **User**

Administrators have the ability to define custom roles and modify existing ones. Details on role management will be discussed further in the chapter "Roles and Permissions."

History Tab

The History tab records all modifications to the user's profile, including the timestamp and the user ID of the person who made the change. The left side of the screen provides options to filter the history by date, user, or other criteria, while the right side displays detailed information about the specific changes made during a transaction.

Two types of Users

Portfolio seamlessly manages two distinct types of users within the same User Management application, bridging PC, Web, and IBM i platforms into a unified solution.

IBM i Users	These users are directly linked to the native IBM i user profile, typically identified by IDs up to 10 characters long. By default, the system is configured to automatically create an IBM i user profile upon the first login attempt. During this process, the system adopts the User ID and Name from the IBM i User Profile. Additional information, such as roles, status, and language, is configured based on a template called "User," which by default assigns the Administrator role. It is recommended to change this default role to a lower privilege level, such as "User," immediately after installation for security reasons. Administrators can also manually create IBM i users and link them to the appropriate IBM i User Profile.
Other Users	Unlike IBM i users, PC and Web users are not tied to an IBM i user profile. Their IDs are more flexible, often based on email addresses. Administrators can

create Web users directly within the Users application, providing flexibility for managing users who primarily access the system via the web interface.

The administrator can effectively manage both types of users through the User Management application, ensuring that access and security are consistently maintained for both IBM i and Web user types.

User Profile Status

The **User Profile status** is essential for managing user access and activity within the system. Administrators can assign one of three status levels to a user profile:

1. **Active (*ENABLED)**: The user has full access to the system, depending on their assigned roles and permissions.
2. **Locked**: This status is generally applied when there are security concerns, such as multiple failed login attempts. The user cannot access the system until the account is unlocked by an administrator.
3. **Inactive (*DISABLED)**: The user is prevented from logging in or using any applications. Access will remain suspended until the profile is re-enabled.

Administrators can modify these statuses by right-clicking on the user and selecting the appropriate status. All changes, including status updates, are logged for auditing purposes.

Special Rules for IBM i Users

For users on the IBM i system, the profile can only be set to ***ENABLED** or ***DISABLED**. The "Locked" status does not exist on IBM i. When a user fails to log in after three attempts (default), Portfolio marks the profile as "Locked," while the profile is set to ***DISABLED** on IBM i.

To ensure proper synchronization between Portfolio and IBM i, the following rules apply:

- Setting a user profile to **Active** in Portfolio will change the profile to ***ENABLED** on IBM i.
- If the user profile is set to ***ENABLED** on IBM i, the status will be updated to **Active** in Portfolio upon the next login.

Status Behavior Comparison

Portfolio Status	IBM i Status	Forgot Password	Login Portfolio	Description
Active	*ENABLED	Yes	Yes	User is active and has full system access.
Active	*DISABLED	No	No	User is *DISABLED on IBM i and cannot log in.
Locked	*ENABLED	Yes	Yes	User failed to log in but is *ENABLED on IBM i. Status will change to Active upon successful login.
Locked	*DISABLED	Yes	No	User failed to log in and must use the "Forgot Password" process to recover access.
Inactive	*ENABLED	Yes	Yes	User is *ENABLED on IBM i but inactive in Portfolio. User can log in.
Inactive	*DISABLED	No	No	User is inactive, and their profile cannot be reactivated without administrator intervention in either Portfolio or IBM i.

Forgotten password

If a user forgets their password, they can obtain a new one through the following methods:

1. Request from login screen

Users who have forgotten their password can request a new one directly from the login screen. After entering their profile ID, they should click on the "Forgot Password" link. This action will trigger an email, sent to the registered email address associated with the profile ID, with the subject "Forgotten Password." The email will contain instructions and a link to a "Change Password" page.

On the "Change Password" page, the user can create a new password. The new password must comply with the system's security requirements; otherwise, the system will reject the change. To ensure accuracy, the user must enter the new password twice—once in the initial field and again in a confirmation field, as the password will not be visible while being typed. Once the new password is successfully created, the user will be redirected back to the login screen.

If the user's profile ID is marked as 'Inactive' or there is no registered email address, the reset email will not be received, and the user must contact an administrator for assistance.

2. **Request a password from the administrator**

Alternatively, users can request a password reset through an administrator. Administrators can access a list of all users via the Users application within the System Settings.

Right-clicking on any user allows the administrator to change the user's password directly or send a password reset request, prompting the user to reset their password using the method described above.

Users can change their password at any time through the user settings menu located in the top-right corner of Portfolio.

Expired password

IBM i user profiles can be configured to expire after a set period, based on the system value **QPWDEXPWNRN**. Users typically receive a warning seven days before the password expiration. When this occurs, Portfolio detects the warning and prompts the user with a pop-up window upon login, requesting a password change. Although users may ignore this request, they must change their password before the expiration date.

If the password expires, the IBM i user profile will be "Disabled," and the Portfolio profile will be "Locked." In such cases, users can follow the "Forgot password" procedure to reactivate and enable their profile.

The password expiration interval can also be customized individually for each user profile by setting the **PWDEXPITV** keyword to any value between 1 and 366 days, or to ***NOMAX** (no expiration). Alternatively, it can be set to the default system value ***SYSVAL**.

Password set to expire

When creating new profiles, the Administrator has the option to set the password to expire using the keyword **PWDEXP**. This ensures that when the user logs in for the first time, they are required to change their password. This process guarantees that only the user knows their password.

Maintain Email Templates and Language

The “Password Reset” and “Forgotten Password” emails can be further customized by modifying the templates located at:

```
\www\iceapp\<Portfolio Folder>\customized\templates\setPassword\emails\
```

To adjust the language and wording of these emails, you can use the “Translations” application found in the System Settings menu.

User Roles and Permissions

In the Portfolio system, effective management of roles and permissions is crucial for maintaining a secure and organized environment. Roles define the access levels and functionalities available to different users, ensuring that each user has the appropriate level of access according to their responsibilities. Permissions, on the other hand, are the specific rights granted to these roles, determining what actions users can perform within the system.

This chapter provides a comprehensive overview of how to create, manage, and assign roles and permissions. It will guide you through the default roles provided by the system, the process of creating custom roles tailored to specific needs, and how to assign the appropriate permissions to ensure that users have access only to the necessary features and data.

Roles

When you create a user, it is essential to assign appropriate roles to ensure they have access to the necessary applications and functions within the system. Each user can be assigned one or multiple roles, depending on their responsibilities.

To manage roles within the system, navigate to the System Settings and select "Roles". You will be presented with a grid displaying all the roles currently defined in the system.

Default Roles

By default, the system comes with three predefined roles:

Admin	The administrator role, which has full access to all areas and functions within the system.
Superuser	This role has access close to that of an Admin but with some restrictions in the System Settings.
User	A standard user role with limited access, typically confined to the applications and functions necessary for routine operations.

Creating a New Role

There may be instances where a new role needs to be created to accommodate specific requirements or permissions. To create a new role, follow these steps:

1. **Create**

Begin the process by clicking on the "Create" button. This will open a new screen with multiple tabs at the top.

2. **Content Tab**

This tab have only three fields:

- **Type** - Enter a short name for the new role.
- **Description** - Provide a more detailed description of the role's purpose.
- **Status** - Set the status of the role, which can either be active or inactive.

3. **Permissions Tab**

The Permissions tab presents a tree structure that organizes the various permissions available within the system. A permission is defined as the authorization to perform specific actions, such as accessing an application, executing a function, or performing operations on data, including reading, writing, updating, or deleting records in a file.

- **Left Panel** - Shows all permissions that exist within the system.
- **Right Panel** - Shows the permissions that have been assigned to the role.

Permissions are rarely used in conjunction with IceCap, as the authorization system on IBM i cannot and should not be overridden.

4. **Menu Tab**

The Menu tab enables you to include or exclude applications from a role by using checkmarks at each level of the menu structure. This view offers a valuable overview of the role's access to every application within the system. When access is granted to an application located several levels down in the menu structure, the role will automatically gain access to all levels above it. In practice, this means that it is not necessary to explicitly grant access to parent folders or higher levels in the menu hierarchy.

5. Save

After completing all necessary configurations, click "Save" to create the new role. The role will now be available for assignment in the user management system.

This process ensures that roles are created and managed in a structured manner, allowing for clear and precise control over user access and permissions within the IceCap and Portfolio system.

Permissions

Setting up permissions within the Portfolio system is essential for controlling user access to various functions and data. Permissions determine what users can view, modify, or execute within the system. These permissions are assigned to roles instead of individual users, which enhances flexibility and security in managing access.

When a user is assigned multiple roles, the permissions are cumulative. In this structure, duplicate permissions do not exist; instead, the user gains the total set of permissions across all assigned roles.

Permissions are displayed in a hierarchical tree structure. The default hierarchy includes the following levels:

- Applications
 - Systems
 - Data
 - Copy
 - Create
 - Delete
 - Export
 - Read
 - Update
 - Function
 - Columns
 - Filters

This structure is utilized, among other things, to define permissions within applications under System Settings. Applications built using the Sitemule Model Studio automatically integrate with this permission structure due to their unified design and functionality. For other applications, this structure can be adopted or customized as needed.

Permissions are binary (true/false) and are accessed through specific paths, such as:

```
app.corpdata.data.export: true  
app.corpdata.data.delete: false  
app.corpdata.functions.columns.arrange: false  
app.corpdata.functions.filter.use: true
```

In the example above, the role associated with these permissions can filter and export data in the grid but cannot delete data or rearrange columns.

When creating a new permission, administrators can set a default value (true or false), guiding the system setup and making the configuration process more efficient.

Customizing Portfolio

In Portfolio, themes allow you to personalize the visual appearance of your IceCap and Portfolio environment, enhancing the user experience and ensuring alignment with your organization's branding. Themes provide flexibility to adjust colors, styles, and layouts, creating a more engaging and tailored interface.

Portfolio comes with several predefined themes, including both light and dark scheme options, applicable across the entire application. You can also integrate your company's logo, customize the login screen, and set system names to reflect your corporate identity. Implementing themes in Portfolio is straightforward, offering a powerful way to customize the look and feel of your applications, making them more visually appealing and user-friendly.

Themes

Portfolio is delivered with eight predefined themes, three of which are in dark scheme:

Theme	Description	Color scheme
Daylight	Light blue with a fresh attitude (Default)	Light
Limelight	Green with an environment look	Light
Sunlight	Red with a warm atmosphere	Light
Skylight	Blue with a cool business format	Light
Twilight	Dark Grey with a high contrast	Light
Black Op	Dark Black for light-sensitive eyes	Dark
Deep Blue	Dark Black with a blue tone	Dark
Classic 5250	Black & Green with a touch of legacy	Dark

In the "Classic 5250" theme, all IceCap screens are displayed with colors that closely resemble the legacy 5250 black and green terminal, while the rest of Portfolio and other applications are shown with the "Deep Blue" theme.

To change the theme, navigate to the User Settings in the Navigation Top Panel. Changes take effect immediately and will be saved for future sessions.

Company Logo and Login background

You can customize the Login Screen and Navigation Panel in Portfolio by adding your company logo, setting it as a favicon, and customizing the background image on the Login Screen to align with your company's branding. The images used must adhere to the following specifications:

Image	Description	Format	Recommended size in pixels
topbar-logo.png	Logo on Navigation Panel	PNG/SVG	32-96 x 32-48
logo.png	Logo on Login Screen	PNG/SVG	64-256 x 64
bg-logo.jpg	Background for Login Screen	JPG/SVG	1920 x 1080 to 3840 x 2160
favicon.svg	Favicon	SVG	Resizeable

Important Guidelines:

- **Transparent background** - Ensure that PNG/SVG files have a transparent background to allow seamless integration with any background.
- **Aspect ratio:** Background images should maintain a 16:9 aspect ratio (width) for optimal display across various devices.
- **Image conversion:** If image format conversion is necessary (e.g., PNG to SVG), using tools like [Convertio](#) is recommended to preserve quality.
- **Image resolution:** Use images with a resolution equal to or higher than the final output to maintain high-quality results after conversion.

These images should be placed in the following IFS directory:

```
/www/iceapp/<Portfolio Folder>/images/
```

Before replacing the default images, make sure to create a backup. After updating the images, clear your browser cache or use the shortcut **CTRL-SHIFT-R** to ensure the changes take effect.

Customizing the setup

Further customization of your Portfolio environment can be achieved through the system settings file located in the IFS directory:

```
\www\iceapp\<Portfolio Folder>\customized\system-settings.js
```

This file is structured into three key sections:

1. Login Screen Settings

Within the `System.login.Settings` section, you have the ability to modify the text labels on the login screen, add translations, and adjust the naming, format, and placement of images.

```
Ext.define("System.login.Settings", {
    override: "Portfolio.login.view.Template",

    getLoginConfig: function() {

        return {
            loginText: ip2.i18n({
                da: "Login",
                en: "Login"
            }),
            forgotText: ip2.i18n({
                da: "Glemt kodeord",
                en: "Forgot password"
            }),
            logo: "/portfolio/images/logo.png",
            backgroundImage: "/portfolio/images/bg-login.jpg",
            buttonBackgroundColor: "#fff",
            buttonTextColor: "#000",
            forgotTextColor: "#e7e7e7",
            color: "#fff",
            errorColor: "red",
            logoStyle: {},
            inputs: [
                {
                    id: "userid",
                    type: "text",
                    label: ip2.i18n({
                        da: "Bruger ID",
                        en: "User ID"
                    })
                }
            ]
        };
    }
});
```

```
        {
            id: "password",
            type: "password",
            label: ip2.i18n({
                da: "Kodeord",
                en: "Password"
            })
        }
    ]
};
}
```

2. Top Navigation Panel Settings

The `System.top.Settings` section allows you to modify the visual behavior of the profile icon, customize the logo, and add a system name to the Navigation Panel.

- **Change the Profile Icon:** By default, a profile icon is displayed until the user replaces it with an avatar or photo. You can opt to replace this icon with the user's Profile ID by setting `showIcon:false`. This change will display the User ID in a circle instead of the icon. Additionally, enabling `showTip:true` will display the full name of the user as a tooltip.
- **Change the Logo:** Replace the default Portfolio logo with your company logo by specifying the URL of the new logo and adjusting the width and height as needed:
`logo:{url:"my-logo.png",width:128,height:32}.`
- **Add a System Name:** It is recommended to change the system name from the default "Portfolio" to a name that is familiar to users within your company. For organizations with multiple environments, consider appending stages like "Development," "Test," or "Production" to the name. This helps users quickly identify the environment they are working in. For example: `text:"My System, Development"` and activate it by setting `showText:true`.

```
Ext.define("System.top.Settings", {
    override: "Portfolio.top.view.Main",

    getTopConfig: function() {

        return {
            profile: {
                showIcon: true,
                showTip: true
            },
        },
    },
});
```

```
        text: "",
        showText: false,
        logo: {
            url: "",
            width: 32,
            height: 32
        }
    };
}
});
```

3. Portfolio Settings

In the `System.portfolio.Application` section in `\www\iceapp\<Portfolio Folder>\customized\system-settings.js`, you can modify the Favicon (favorite icon) and the Title that appears in the browser tab, bookmark bar, and other web locations. This icon serves as a visual identifier for your site. Additionally, you can change the title from the default "Portfolio" to your system name, matching the name used in the Navigation Panel.

```
Ext.define("System.portfolio.Application", {
    override: "Portfolio.Application",

    getAppConfig: function() {
        return {
            icon: "",
            title: "Portfolio"
        };
    }
});
```

These changes will take effect the next time users log in.

Portfolio Client Manager

The Portfolio Client Manager is an application installed natively on computers running Windows, Linux, or MacOS. It is designed to improve user experience by running the standard Portfolio in an embedded browser, presenting it as a native application across different platforms.

Key Features

The Portfolio Client Manager provide the following essential features:

- **Enhanced Browser Controls:** Allows more granular control over browser behavior within the Portfolio Client.
- **Management of Multiple Environments:** Enables users to configure and manage various environments for different clients, applications, or work contexts.
- **Bridging between the Client and IBM i:** Facilitates direct communication and integration with IBM i systems, ensuring seamless data sharing and process execution.

Portfolio Client

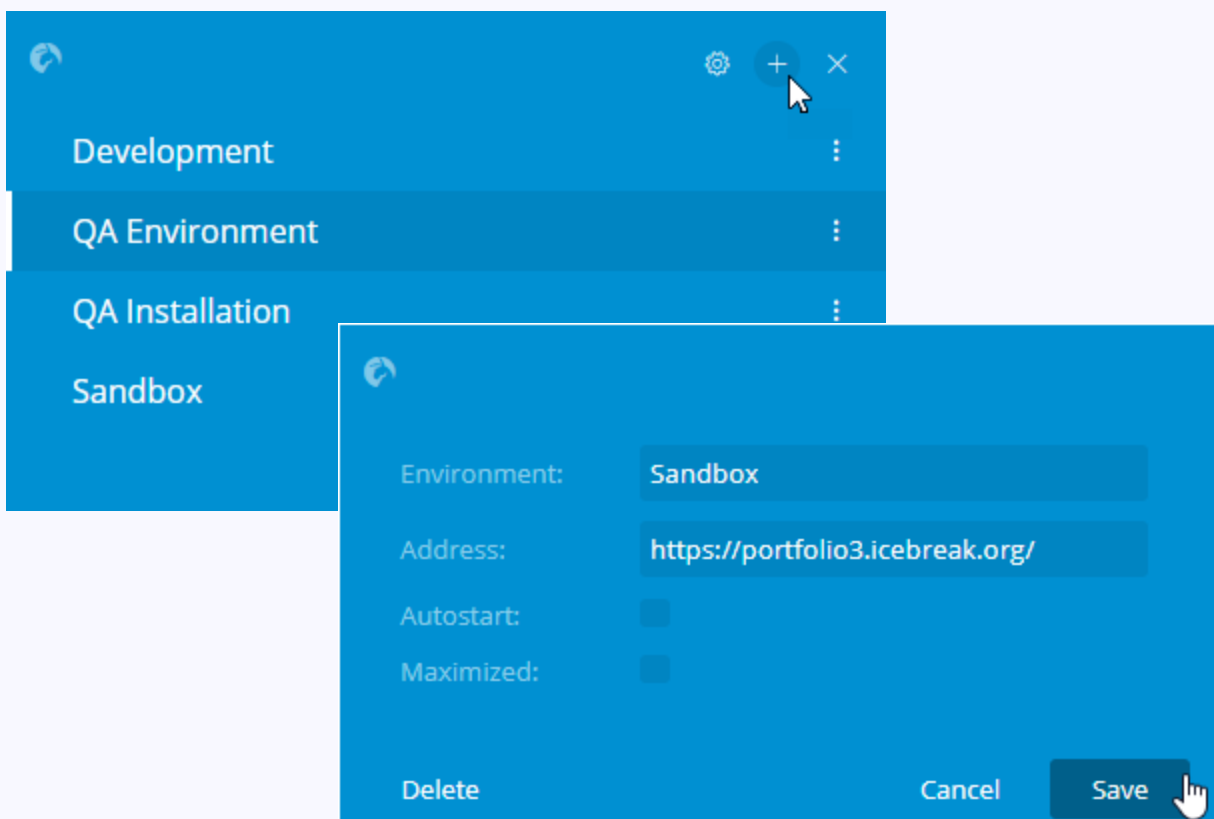
The Portfolio Client enhances the usability of the Portfolio environment with the following features:

- **Automatic Zoom Adjustment:** The Client dynamically adjusts the zoom level of content to the optimal size, and recalibrates the zoom percentage when the window is moved to a screen with a different resolution. This feature is particularly important when IceCap presents 5250 screens formatted as 27 lines with 132 characters per line in Flex, Flow, or Flip modes. It is especially beneficial on screens with a resolution of 1920x1080 pixels or lower and when text size is increased (e.g., 125%). In situations where traditional web browsers would introduce horizontal and vertical scrollbars due to a combination of low screen resolution and increased text size, the Client ensures that content is displayed optimally without scrollbars.
- **Tab Request Interception:** The Client can intercept requests that would typically open in a new browser tab and redirect them to open in a new tab or window within the Portfolio Application Panel. This feature helps integrate documents and images further into the Portfolio environment.

- **Notification Center Integration:** The Client can send messages, such as break messages from the 5250 environment, directly to the computer's notification center, improving system integration.
- **Application Launch from IBM i:** The Client enables launching any application from the 5250 environment or other web applications running in Portfolio, bridging functionality between IBM i and modern systems.

Portfolio Client Manager

The Manager is a background service program that manages and launches environments for clients efficiently. It is easily accessible from the system tray, allowing quick interaction and control over various environments.



Setting up the Manager

The Manager offers two primary settings for startup behavior:

- **Starts with system:** The Manager will automatically launch when the user logs in.
- **Starts hidden:** The Manager will start minimized in the system tray.

Setting up a New Environment

To create a new environment:

1. Open the Manager and click the Add (+) button.
2. Fill out the environment setup form:
 - **Environment:** Name of the environment.
 - **Address:** URL or IP address, including the port number.
 - **Autostart:** Enables the environment to launch automatically when the Manager starts.
 - **Maximized:** Configures the environment to open in a maximized window.
3. Save the setup, which will be stored in a JSON file.

The Manager supports any web application or website, making it suitable as a shared hub for organizational environments.

Configuration Files

The Manager stores configuration files in the following directory:

C:\Users\<YourProfile>\AppData\Local\Portfolio Environment Manager\

- **Manager Settings:** Stored in `settings.json`:

```
{
  "autostart": true,
  "minimized": false,
  "editable": true
}
```

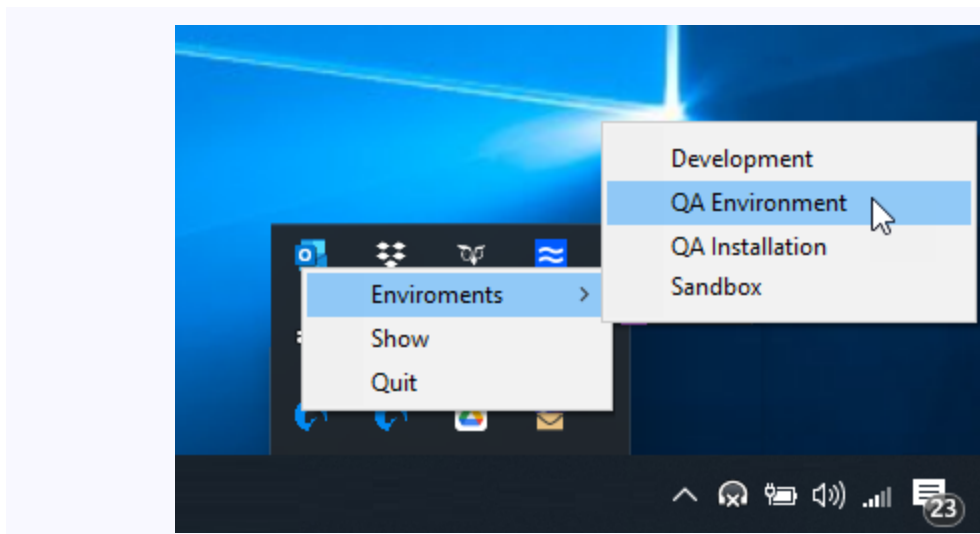
The "editable" setting can only be changed within the configuration file. Setting "editable": false prevents the user from modifying settings or creating new environments.

- **Environment Settings:** Stored in `servers.json`:

```
[
  {
    "id": "1716383667266",
    "title": "QA Environment",
    "src": "http://dksrv206:36700",
    "start": false,
    "maximized": false
  },
  {
    "id": "1716383882785",
    "title": "Sandbox",
    "src": "https://portfolio3.icebreak.org/",
    "start": false,
    "maximized": false
  }
]
```

The order of environments can be changed by rearranging the sections in `servers.json`. However, the `"id"` is a unique identifier and must not be altered.

The files can be distributed to users as predefined, non-editable settings.



Replacing the IBM Access Client Solution (ACS)

The Portfolio Client Manager are designed to replace the IBM Access Client Solution (ACS), which traditionally handled communication between computers and IBM i servers. This bridge was crucial

for opening files on the Integrated File System (IFS) and was a key component of PC Organizer functionality.

With the Portfolio Client Manager installed, it can replace the ACS, providing the same bridge between the computer and IBM i. This allows for similar functionality, even when using a standard web browser running Portfolio.

Installation

Requirements

Ensure the following requirements are met before installation:

- Windows 10 or newer (Beta versions for macOS ?? or newer and Linux ?? or newer)
- Administrator rights on the computer

Note: The installation does not require access to IBM i, nor do IceCap and Portfolio need to be installed on IBM i.

Installing

To install the Portfolio Client Manager, follow these steps:

1. Download the latest version from: <https://webfiles.system-method.com/download/portfolio/>
2. Identify the correct installer based on your operating system:
 - Windows: **.msi** extension
 - Linux: **.deb** extension
 - macOS: **.dmg** extension
3. Run the appropriate installer and follow the instructions in the installation wizard.

Uninstalling

To uninstall, follow the standard uninstallation procedure for your operating system.

The Notification Center

The Notification Center is a crucial feature of the Portfolio environments, facilitating both internal user communication and system-to-user notifications. It is designed to enhance communication efficiency by delivering updates about system processes, background jobs, and application error messages directly to users.

Receiving Messages

When a user receives a message from the system or another user, a small notification box briefly appears in the lower right corner of the screen. This box automatically disappears after a few seconds. Users can access all sent and received messages by opening the Notification Center, which is located within the User Settings Menu in the upper right corner of the interface.

To view the Notification Center, click on the User Settings Menu. The Notification panel will slide in from the right side of the screen, displaying a list of all sent and received messages. These messages can be expanded or collapsed using the arrow located in the lower right corner of each message. Users can also delete individual messages by clicking the "X" in the upper right corner of the message box.

Sending Messages

To send a message, users can utilize the Notification panel. By selecting the recipient and typing a short text, users can send basic messages. The messaging area within the Notification panel can also be expanded into a separate window, providing additional features such as a subject line and a fully equipped text editor. This expanded area is ideal for composing more detailed or complex messages.

Messages can be sent to one or more users. When selecting a recipient, a list of all Portfolio users is displayed. Active users are marked with a green dot, indicating their availability. To select a recipient, click on their name, and they will be added to the recipient field. Recipients can be removed by clicking the "X" next to the recipient's name in the field. If a recipient is inactive when the message is sent, they will receive the notification upon their next login.

Integration with System Notifications

The Notification Center can be integrated with the computer's native notification system, allowing messages to appear as native system notifications even when the Portfolio application is minimized. This ensures that users receive critical updates regardless of the application's state. Integration can be enabled by installing the Portfolio Client Manager or by configuring Portfolio to run securely under HTTPS.

Configuration

IceCap offers extensive customization options, primarily through the Emulation and Navigation components, which are designed to meet specific user and system requirements. The customization areas include:

Emulation	• Interpretation: Controls how data streams from the IBM i system are interpreted and displayed. • Presentation: Defines the layout and appearance of 5250 screens within the IceCap environment. • Translation: Manages the translation of screen elements into different languages, ensuring multilingual support.
Navigation	• Recognition: Facilitates the automatic detection of screens and workflows to enhance user navigation. • Configuration (F-keys, Options): Customizes the functionality of F-keys and option handling across applications.

Plugins and Customization Templates

IceCap is delivered with default templates, referred to as Plugins, for each area of customization. These Plugins can be found in the IFS directory:

```
/www/iceapp/IceCap2/plugins
```

By default, these Plugins are tailored to fit the IBM i Operating System and systems that adhere to the IBM Systems Application Architecture (SAA). However, as different systems often have variations that cannot coexist within the same program, components may need to be copied, modified, and multiplied to accommodate system-specific requirements.

Refer to the chapter “Plugins” for more detailed information.

IceCap Configuration

The main configuration changes for IceCap are made in the **IceCap configuration file**, located here:

```
/www/iceapp/IceCap2/icecap_config.xml
```

In environments where IceCap is installed multiple times, there might be several instances of the `icecap_config.xml` file on the system. If there is any uncertainty about which configuration file is active, the IceBreak Server Configuration can be used to identify the correct file for the IceBreak server on which Portfolio is running.

```
<?xml version="1.0" encoding="UTF-8" ?>
<config>
  <default>    <!-- Default config. All tags can be override -->

    <!-- For default general theme. Look in folder: /www/system/styles/ext/xtheme*
(extention .css) -->
    <!-- For default IceCap theme. Look in xml file: /www/icecap/icecap_theme.xml -->

  <defaults
    trace="yes"
    tracePath="/www/iceapp/icecap2/trace_flat/"
    logClientActions="no"
    contentLocation="east"
    timeOut="60"
    detectWindows="always"

    showTitle="yes"
    showOptionText="no"
    showOptionField="no"
    showOptionColumn="yes"
    showOptionNo="yes"
    showOptionSeparator="="
    defaultOptions="2,1,5"
    alternativeOptions=""
```

Applying Configuration Changes

Please note that any changes made to the `icecap_config.xml` file will only take effect after the IceBreak server is restarted.

IceBreak Server Configuration

Each IceBreak server has its own **webconfig.xml** file, located at:

```
/www/iceapp/icebreak/<Server Name>/webbconfig.xml
```

To identify which IceCap configuration file is currently in use, search for "**icecap_config**" within the Portfolio's **webconfig.xml** file. This search will provide the exact file path and name of the configuration file used by the server.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

    <upload>
        <map alias="packages" path="packages" />
        <map alias="*" path="/tmp" />
    </upload>

    <contentEncoding
        type="GZIP"
        threshold="10"
        dontCompress="gif,jpeg,jpg,png"
    />

    <!--
        The response content is by default windows-1252 for backward compatability reasons
        only. We suggest that you change it to UTF-8
        and use UTF-8 for both your source and templates. If you entire system is in UTF-8
        you can even gain
        a performance boost by setting the "serveAlwaysUTF8" to true
        NOTE: programs that use "setContentType(...)" overrides the defaultCharset at
        runtime.
    -->
    <content
        defaultCharset="utf-8"
        serveAlwaysUTF8="true"
        serveAlwaysChunks="true"
    />
```

Presentation keywords

The Presentation Keywords in IceCap allows you to customize the appearance and behavior of your IBM i applications, transforming traditional 5250 screens into modern web-based interfaces. These keywords are configurable settings that control various aspects of the user interface, such as display elements, navigation, and interaction.

- 1. **Customization** - Adjust the presentation of screens by defining how fields, labels, and other elements are displayed. This includes changing colors, fonts, and layouts to create a more user-friendly and visually appealing interface.
- 2. **Enhanced Usability** - Improve the user experience by configuring keywords that streamline navigation, such as enabling tooltips, displaying function keys as buttons, and organizing fields in a logical order.
- 3. **Flexibility** - Apply different presentation modes (Flat, Flex, Flow, Flip) to suit various user needs and preferences. Each mode offers distinct customization options that can be tailored to specific workflows and tasks.
- 4. **Consistency** - Ensure a consistent look and feel across all applications by standardizing the use of presentation keywords. This helps maintain a cohesive user experience, reducing the learning curve for new users.

Using presentation keywords in IceCap, administrators can easily transform and modernize their IBM i applications, enhancing both functionality and user satisfaction.

Presentation keywords			Mode			
Keyword	Default	Description	Flat	Flex	Flow	Flip
showTitle	"yes"	The keyword controls the title's visibility only on the 5250 screen and does not affect the title shown in the Portfolio tab, which remains visible.	-	-	-	-
showTitleInTab	"yes"	The 5250 screen title will be shown in the Portfolio tab otherwise the menu title will be used.	-	-	-	"no"
showOptionText	"no"	Deprecated				
showOptionField	"no"	Deprecated (Default is misleading – use showOptionColumn)				
showOptionColumn	"yes"	Shows the option column in the grid	n/a	n/a	-	"no"

Presentation keywords			Mode			
Keyword	Default	Description	Flat	Flex	Flow	Flip
showOptionNo	"yes"	Shows the option number in the option menu	n/a	n/a	-	"no"
showOptionSeparator	"="	Inserts the separator character between the option # and the description	-	-	"_"	" "
defaultOptions	"2,1,5"	The default (dobbelt-click) option is found in the prioritized order	n/a	n/a	-	-
alternativeOptions	""	Include alphabetic options like X=Select and C=Change as "X,C" etc.	-	-	-	-
showAllKeys	"no"	Shows all F-keys	-	"yes"	-	-
showKeysText	"no"	Shows F-key on screen as text	-	-	-	-
showKeysAsButtons	"yes"	Shows F-keys as Buttons	-	-	-	-
showKeysAsTabs	"no"	Shows F-keys as Tabs. Predefined F-keys will still be shown as buttons.	-	-	-	"yes"
showKeysAsPredefined	""	F-keys will be added to the list of predefined keys and shown as buttons.	-	-	-	"yes"
showKeysFxx	"yes"	Shows F-key number	-	-	-	"no"
showKeysToolTip	"no"	Shows tooltips on F-keys	-	-	"yes"	"yes"
showKeySeparator	"="	Inserts the separator character between the key # and the description	-	"_"	" "	" "
showKeysMenu	"no"	Shows all F-keys in a menu	-	-	"yes"	-
showStatusBar	"yes"	Shows the Status Bar with position, language, mode, and stage	-	-	-	"no"
showPageUpDown	"no"	Shows an arrow for PageUp and PageDown in the Grid	n/a	n/a	-	"yes"
showLabelUnderlined	"no"	Shows field labels with an underline and removes all tailing dots	-	"yes"	"yes"	"yes"

Presentation keywords			Mode			
Keyword	Default	Description	Flat	Flex	Flow	Flip
showLabelAbbreviationDot	"no"	Show dot that might be from an abbreviation of a word	-	-	-	-
showLineOneScreen	"yes"	Shows the first line on screen	"yes"	"yes"	"yes"	"no"
showLineOneWindow	"yes"	Shows the first line in window	"yes"	"yes"	"yes"	"no"
showEnterKey	"no"	Enter will be shown as a key on every screen	-	-	"yes"	"yes"
enterKeyText	"OK"	Enter will be named to this - if Enter isn't already named on the screen	-	-	"OK"	-
helpKey	"F1"	Predefine key for Help	-	-	-	-
exitKey	"F3"	Predefine key for Exit	-	-	-	-
promptKey	"F4"	Predefine key for Prompt	-	-	-	-
refreshKey	"F5"	Predefine key for Refresh	-	-	-	-
cancelKey	"F12"	Predefine key for Cancel	-	-	-	-
moreOptionsKey	"F23"	List more Options – if null, search for more Options will not be attempted	n/a	n/a	-	-
moreKeysKey	"F24"	List more F-keys – if null, search for more F-keys will not be attempted	-	-	-	-
messageLine	""	Deprecated				
optionsSplitValue	" "	Character value to separate options in a text string. Example value " , "	-	-	-	-
rightAdjustNumeric	"IO"	Right adjust numeric always (IO) or only when output (O)	-	-	-	"O"
columnHeaderAttribute	"22"	Attribute to identify column heading. Default: White ="22", Green="*None"	-	-	-	-
useInterpretation	"yes"	Use interpretation of texts to apply components	-	"yes"	"yes"	"yes"
useTags	"yes"	Use interpretation of Tags to apply components	-	"yes"	"yes"	"yes"

Presentation keywords			Mode			
Keyword	Default	Description	Flat	Flex	Flow	Flip
useSubfileTransformation	"no"	Subfiles are transformed to Grids	-	-	"yes"	"yes"
useScreenTranslation	"yes"	Activates the Globus icon in the status bar to enable text translation	-	-	-	-
translateToLanguages	"da,en"	Lists possible translation languages	-	-	-	-
continueInWindow	"yes"	Continue operating in a windowed environment despite its full-screen design in 5250	"yes"	"yes"	"yes"	"yes"

Note: Where relevant, the default is based on the IBM SAA Standard used in the OS

Emulation keywords

The primary role of emulation keywords is to manage the data stream from the virtual terminal, converting it into a format that can be easily manipulated and displayed in a web client.

This conversion process involves defining how elements such as screen titles, function keys, and data fields are presented in the web interface. Emulation keywords enable the customization of these elements, allowing for a more intuitive and user-friendly experience.

Emulation keywords		
Keyword	Default	Description
program	"/menu/icecap.aspx	
useHints	"no"	
hintsXMLhive	""	
hintsXMLpath	""	
numberOfSubfilePagesToStack	""	

Plugins keywords

Plugins keywords in IceCap are used to define specific Plugins that can modify the behavior of applications or add new capabilities. These keywords specify the programs and libraries that contain the Plugin logic, ensuring seamless integration with the existing system.

Plugins keywords		
Keyword	Default	Description
navigatorProgram	"*LIBL/*NONE"	
emulatorProgram	"*LIBL/*NONE"	
interpretationProgram	"*LIBL/*NONE"	
pcoProgram	"*LIBL/*NONE"	
i18nXlateProgram	"*LIBL/*NONE"	
xxx_deviceNameProgram	"*LIBL/*NONE"	
xxx_varyoffExitProgram	"*LIBL/*NONE"	

Operation keywords

Operation keywords define how IceCap handles different processes and tasks. For example, they can set parameters for tracing system activities, specifying trace paths, and determining timeout durations for sessions. These settings help in diagnosing issues, monitoring system performance, and ensuring that applications remain responsive and reliable under different conditions.

Using operation keywords, administrators can also customize how IceCap interacts with virtual devices and manages user sessions. This includes configuring the detection of window sessions, setting content locations, and managing client actions logging. Such granular control is essential for maintaining high operational standards and ensuring that the system can adapt to the dynamic needs of the organization.

Operation keywords		
Keyword	Default	Description
trace	"no"	
tracePath	"/www/iceapp/icedev/trace/"	
logClientActions	"no"	
contentLocation	"east"	
timeOut	"60"	
detectWindows	"always"	detectWindows: never,always, native
inviteTimeOut	1000	Timer in milliseconds (1000 = 1 second)
automaticScreenScripting		<automaticScreenScripting> <screen title="Sign Off" action="ENTER" /> <screen title="Display Messages" action="ENTER" /> </automaticScreenScripting>

IBM i Configuration

Typically, you don't need to modify your IBM i configuration for IceCap to function properly. However, understanding how IceCap operates is crucial.

System Values

Whenever IceCap opens a tab in Portfolio, it initiates a new interactive job using a distinct virtual workstation device. Therefore, six open tabs correspond to six virtual devices. When a tab is closed, the associated interactive job ends, and the virtual device is varied off. We do not delete these devices, as they are intended to be reused for subsequent interactive jobs. By default, IceCap prefixes device names with “ICE”.

Given the increased usage of virtual devices, it's advisable to adjust the system value `QAUTOVRT` (Autoconfigure Virtual Devices) to a higher threshold, ideally setting it to the IBM default value `*NOMAX`.

Additionally, users can be restricted to a limited number of device sessions by setting the system value `QLMTDEVSSN` (Limit Device Sessions) or the keyword `LMTDEVSSN` (Limit Device Sessions) on the User Profile. The ideal setting would be IBM's default values `QLMTDEVSSN(0)` and `LMTDEVSSN(*SYSVAL)`. If the user reaches the maximum number of device sessions, the sign-on display will appear instead of the program.

Limit Capabilities

To fully utilize the Portfolio menu system and experience it as a multitasking environment, traditional menus from the 5250 environment must be moved into the visual menu structures in Portfolio. After that, IceCap will sign on as the current user in the background and call the chosen program.

Some users may have “Limited Capabilities” according to the keyword `LMTCPB` on their user profile. Limiting users' capabilities is often done to prevent them from calling programs and commands from a command line. However, this setting will also prevent users from starting programs from the Portfolio menu structure.

To give users with limited capabilities `LMTCPB(*YES)` and `LMTCPB(*PARTIAL)` full access to Portfolio, you need to change the keyword `INLPGM (Initial Program to Call)` to `INLPGM(<IceCapLibrary>/STRICECAP)`. This adjustment will allow the user to start individual programs while maintaining their limited capabilities.

Appendix A - Implementation Process Template

The process

This implementation process represents a typical approach, with the estimated number of days reflecting the average duration for a conventional ERP solution. This template can serve as a guideline for your project or a source of inspiration.

1. **Web-enable the ERP** (Estimate 1-2 days)
 - a. Set up IceCap and Portfolio.
 - b. Synchronize the ERP menu structure.
 - c. Replicate the ERP authorization rules.
 - d. Eliminate or transform legacy infrastructure approaches.
2. **Improve usability** (Estimate 3-4 days)
 - a. Supply visual aids such as combo boxes, checkboxes, and radio buttons on as many fields as possible. Identify fields in general so that they will take effect across every screen.
 - b. Replace “misuse” of fields and weak validation with appropriate field labels and selection lists.
 - c. Optimize data processing with advanced SQL search boxes featuring “Type Ahead” and “Assist” capabilities.
 - d. Ensure data quality through complex field dependency validation.
3. **Increase information** (Estimate 4-5 days)
 - a. Retrieve and display data from other screens to provide relevant information to users.
 - b. Include images, PDFs, and Office documents.
 - c. Provide visual data presentations using charts.
4. **Apply functionality** (Estimate 5-6 days)
 - a. Structure and define the database with the Model Studio in Sitemule Architect.
 - b. Divide the database by domain and import the files into the Model Studio.
 - c. Create new views showing data as spreadsheets with built-in functions like sorting, filtering, grouping, summarizing, and exporting.

- d. Connect views to relevant legacy programs and access them as right-click functions.
5. **Customize workflows** (Estimate 5-6 days)
 - a. Restructure current workflows to be more relevant to users.
 - b. Reuse existing views, forms, and breakouts in new combinations.
6. **Integrate through web services** (Estimate ½-1 day per program)
 - a. Transform programs with IceCap into web services.
 - b. Access the legacy ERP through a unified service layer.
-

The products

The implementation process outlined above relies on the installation of the following products:

- **IceBreak** (Runtime)
- **Portfolio** (Step 1)
- **IceCap** (Steps 1-3, 6)
- **Sitemule Architect** (Steps 4-5)

IceBreak operates in a free runtime version and provides support for the other products. Portfolio is installed separately but is licensed together with IceCap. Sitemule Architect is a licensed product that serves as a platform for developing new applications.

Appendix B - Subfile options & Function keys

IBM developed the [Systems Application Architecture](#) (SAA) as part of their Systems Application Architecture framework, introduced in 1987. A key component of SAA is [Common User Access \(CUA\)](#), which establishes standards for user interfaces to operating systems and applications.

Common User Access (CUA) component, was designed to standardize elements such as screen layout, colors, attributes, and navigation keys. The goal was to provide a uniform experience across different systems and applications, facilitating ease of use for end-users.

Screen Elements	Typical CUA-compliant screens include the following elements: • System Name • Screen Title • Screen Identification • User Profile • Date and Time • Subfiles and Forms • List of Subfile Options • List of Function Keys • Message Line
Text Colors	The IBM 5250 terminal protocol defines seven colors: White, Green, Blue, Red, Pink, Turquoise, and Yellow. In most IBM systems, only the first three colors are used: • White - Screen titles, headlines, column headings, messages, important information • Green - Field labels, field content • Blue - Subfile options, function keys, additional tooltips
Field Attributes	Various attributes help enhance communication with the user: • Reverse - Highlights fields with errors. • Underline - Indicates fields that are open for input

These attributes and colors follow CUA standards, which serve as guidelines, not strict rules. Developers are encouraged to adapt and interpret these standards based on specific application needs.

Subfiles Options

For consistency, we have limited this section to the first nine subfile options, as it is challenging to define a standard usage for the remaining options. The first five options, however, are generally restricted to specific functions.

Option	Name	Description	Comment
1	Select, Create		Standard
2	Change		Standard
3	Copy, Hold		Standard
4	Delete, End		Standard
5	Display, Work		Standard
6	Print, Release		
7	Display more		
8	Work more		
9	Action		

Functions Keys

Out of the 24 available function keys (F-keys), seven have been designated as Standard due to their frequent use. These standard keys are commonly utilized across different systems and are integral to many of IceCap’s functionalities, particularly in Flow and Flip modes.

IceCap’s reliance on these standard function keys ensures a consistent user experience across various applications. If there are any deviations from the standard configuration, such changes

should be properly documented and reflected in the **Icecap_config** file to avoid functional discrepancies.

For further details on configuring F-keys in IceCap, refer to the sections “Presentation keywords” in the “Configuration” chapter.

F-Key	Name	Function	Standard
F1	Help	Action	Standard
F2	Extended Help		
F3	Exit	Action	Standard
F4	Prompt	Action	Standard
F5	Refresh	Action	Standard
F6	Create	Action	
F7	Search		
F8	Undo Retrieve	Action	
F9	Retrieve	Action	
F10	Investigate		
F11	Toggle View		
F12	Cancel	Action	Standard
F13	Assist, Defaults		
F14	Options		
F15	Services		
F16	Find, Repeat		
F17	Top, Position, Subset		
F18	Bottom		
F19	Left, Next		
F20	Right, Subset		
F21	Print, Assistance Level		
F22	More info		

F-Key	Name	Function	Standard
F23	More options		Standard, if many options
F24	More keys		Standard, if many keys

Appendix C - Keyboard shortcuts

We strive to enhance efficiency and enable a mouseless UI by utilizing shortcuts through CTRL and ALT keys. Since IceCap and Portfolio run in a browser, many common browser-related keystrokes are supported.

This summary highlights the most commonly used shortcuts and those we have added.

Windows and tabs

Shortcut	Description
ALT 0-9	Select tab
ALT ½	Cycles through tabs
ALT SHIFT ½	Cycles through tabs backward
ALT Z	Cycles through open windows
ALT W	Tabs to windows
ALT T	Windows to tabs
ALT F3	Closes current tab

Forms

Shortcut	Description
CTRL S	Save changes in the Form
ALT S	Save changes and exit the Form
ESC	Exits the Form

Operations

Shortcut	Description
CTRL A	Select all content

CTRL C	Copy the selected content to the Clipboard
CTRL SHIFT C	Copy and add to the content of the Clipboard
CTRL E	Copy the selected content to the Clipboard prepared for Excel
CTRL SHIFT E	Copy and add to the content of the Clipboard prepared for Excel
CTRL X	Cut the selected content to the Clipboard
CTRL V	Paste the contents of the Clipboard
CTRL Z	Undo an action
CTRL Y	Redo an action
CTRL F	Open Function Keys Menu
CTRL SPACE	Open Options Menu
CTRL HOME	Display Cursor ruler

System Control

Shortcut	Description
CTRL +	Zoom in making the content larger
CTRL -	Zoom out making the content smaller
CTRL 0	Zoom level back to 100% resetting the content size
CTRL SHIFT I	Open Developer Tools
CTRL SHIFT R	Hard Refresh. Reloads the system while ignoring cached content

Customized Shortcuts

Frequently used applications can be assigned shortcuts, allowing users to launch them directly without navigating the menu system. Shortcuts also simplify navigation through top menus and sidebar menus. As an example, we have already assigned shortcuts to a few applications and menus::

Shortcut	Description
ALT A	Open the Application top menu
ALT Y	Open the System top menu
ALT U	Open the User Application
ALT M	Open the Menu Application

Ensure the assigned shortcuts are unique within the sidebar menu structure to function correctly. You can change or remove the above shortcuts if they conflict with your plans for custom assignments.

The screenshot shows a dialog box titled "16 - Edit - Applications". It has two tabs: "Content" and "Roles". The "Content" tab is active. Under the "Info" section, there are fields for "Titel:" (containing "<u>A</u>ppllications"), "Icon:" (containing a folder icon and "pf-windows"), "System:" (empty), and "Url:" (empty). There is a "Searchable:" checkbox which is checked. Under the "Shortcut" section, there are checkboxes for "Alt:" (checked), "Ctrl:" (unchecked), and "Shift:" (unchecked). There is a "Key:" field containing the letter "A". A "Cancel" button is located at the bottom right of the dialog.

This example illustrates how to add a shortcut to a menu and underline the corresponding letter used in the title.

Appendix D - Virtual Terminal APIs

This appendix describes the currently available Virtual Terminal (VT) APIs that can be used in the Emulation, Interpretation, and Navigation components. Note that the VT commands cannot be used in the `IceCap_config.xml`.

IceCap is continually updated with new services. To find an updated list of available APIs, use the following command and press Enter until you see the Procedure list:

```
DSPSRVPGM SRVPGM(<IceCapLibrary>/ICECAPTERM)
```

To retrieve the full definition and the RPG Free syntax, copy and paste from this source:

```
STRSEU SRCFILE(<IceCapLibrary>/QRPGLEREF) SRCMBR(ICECAPTERM) OPTION(2)
```

Deprecated services are marked in red. Please avoid using them and convert to the new services where applicable.

For more detailed information and use cases refer to the “Virtual Terminal” chapter.

Virtual Terminal API	Description
vtClose	When done using the session - always remember to close the session, which also frees up resources.
vtDetectWin	
vtDeviceStatus	Util: returns the device status: Value Definition 0 VARIED OFF The system is not using the description. 10 VARY OFF PENDING The description is being varied off. During this time the system may be taking down the tasks which managed the resource, and so on. 20 VARY ON PENDING The description is being varied on. During this time the system may be putting tasks in place to manage the resource, downloading a program to an I/O processor, communicating with data circuit-terminating equipment (DCE), and so on. 30 VARIED ON The tasks that manage the network interface, network server, line, controller, or device have been put in place by the system, and the system has the capability to communicate with it. 32 VARY ON/CNN PENDING

Virtual Terminal API	Description
	The first of a pair of OptiConnect controllers is varied on but its attached device is not yet in a varied on state. 40 CONNECT PENDING This status is only valid for switched SDLC, IDLC, BSC, or asynchronous lines. The line is in this status while waiting for the switched connection to be established; this can be either a dial or an answer connection. 50 SIGNON DISPLAY This status is only valid for display devices. The system is preparing the device to receive the sign-on display, or sending the sign-on display, or the actual sign-on display is at the display station. 51 ACTIVE/CNN PENDING The first of a pair of OptiConnect controllers and its attached device are varied on and waiting for the OptiConnect path to be established. 60 ACTIVE The object is successfully placed in VARIED ON status. In addition: for network interfaces and the network servers, one or more attached lines is in a VARY ON PENDING status or higher; for lines, one or more attached controllers is in a VARY ON PENDING status or higher; for controllers, one or more attached devices is in a VARY ON PENDING status or higher; for devices, active status varies depending on the type of device — more information is in the Communications Configuration book, SC41-5401 book. A display device which is at a second sign-on display as a result of pressing the system request key will be considered ACTIVE. 63 ACTIVE READER The device is in use by a spool reader. 66 ACTIVE WRITER The device is in use by a spool writer. 67 AVAILABLE The independent auxiliary storage pool (ASP) device is available for use without function restrictions. 70 HELD This status is only valid for Device Descriptions. The user or the system held the communications device to prevent it from communicating. The Release Communications Device (RLSCMNDEV) command can be used to release the device. 80 RCYPND Error recovery is pending for the line, controller, or device. A message indicating what error occurred appears on the QSYSOPR message queue. 90 RYCYNL Error recovery is canceled for the network interface, line, controller, or device. An error occurred, and the operator replied with a C (to cancel error recovery) to a message, or the operator used a command (ENDNWIRCY, ENDLINRCY, ENDCTLCY, ENDDEVRCY) to end error recovery. 95 SYSTEM REQUEST

Virtual Terminal API	Description
	The display device has been requested by the system and its associated job has been suspended. This occurs as a result of a user pressing the System Request key. 100 FAILED An error occurred for the network interface, network server, line, controller, or device that can be recovered only by varying off and on again. 103 FAILED READER An error occurred for the device while in use by a spool reader. 106 FAILED WRITER An error occurred for the device while in use by a spool writer. 107 SHUTDOWN The NWSD was shut down using an Application Program Interface (API). 110 DIAGNOSTIC MODE The network interface, network server, line, controller, or device resource is being used by problem analysis procedures to diagnose problems, and the resource cannot be used by other users. 111 DAMAGED The network interface, network server, line, controller, or device description is damaged. This is a system error condition. Information indicating when this damage occurred appears in the history log (QHST). Further information may be in the vertical licensed internal code (VLIC) logs. The description must be deleted and created once more before it can be used again. 112 LOCKED The actual status of the resource cannot be determined because another job has an exclusive lock on the description. Retry at a later time, or use the Work With Object Lock (WRKOBJLCK) command to determine which job has the lock on the description. 113 UNKNOWN The status indicator of the description cannot be determined. This is a system error condition. Use the Dump Object (DMPOBJ) command to dump the contents or attributes of the description to a spooled printer file, and contact your IBM representative.
vtDump	
vtFfwDataType	Field Format Word - Date Type Field Shift/Edit Specification - unmapped: 0: VT_FFW_ALPHA_SHIFT 1: VT_FFW_ALPHA_ONLY 2: VT_FFW_NUMERIC_SHIFT 3: VT_FFW_NUMERIC_ONLY 4: VT_FFW_KATAKANA_SHIFT 5: VT_FFW_DIGITS_ONLY

Virtual Terminal API	Description
	6: VT_FFW_INPUT_INHIBIT 7: VT_FFW_SIGNED_NUMERIC
vtFfwDDSDataType	DDS data type S: Signed numeric N: Numeric shift Y: Numeric only D: Digits only
vtFfwIsAutoEnter	FFW: Auto Enter
vtFfwIsDupkey	FFW: Dup Key
vtFfwIsFieldExitRequired	FFW: Field Exit Required
vtFfwIsMandatoryEnter	FFW: Mandatory Enter
vtFfwIsMandatoryFill	FFW: Mandatory Fill
vtFfwIsModifiedDataTag	FFW: Modified Data Tag
vtFfwIsNoAjust	FFW: No Ajust
vtFfwIsProtected	FFW: Protected
vtFfwIsReserved	FFW: Reserved
vtFfwIsRightAjustBlankFill	FFW: Adjust Blank Fill
vtFfwIsRightAjustZeroFill	FFW: Adjust Zero Fill
vtFfwIsUppercase	FFW: Uppercase
vtGetField	Locates an input field and returns its current value. The field value, attributes, type, formatting, etc., are returned in the vtFieldDS.
vtGetFieldLoc	Locates an input field and returns its current value. The field value, attributes, type, formatting, etc., are returned in the vtFieldDS.
vtGetFieldNo	The field value, attributes, type, formatting etc is returned in the vtFieldDS.

Virtual Terminal API	Description
vtGetJSON	Returns the current panel as a JSON object. Deprecated: Use individual: vtGetJsonScreenInfo, vtGetJsonInputFields, vtGetJsonOutputFields
vtGetJsonInputFields	Returns the current panel array of input fields as JSON objects.
vtGetJsonOutputFields	Returns the current panel array of input fields as JSON objects.
vtGetJsonScreenInfo	Returns the current panel base info as JSON objects.
vtGetOutputFormatList	Return the list of formats on the screen.
vtGetOutputList	Return the list of output data for the current window / of screen.
vtGetScreen	Returns a substring of the current panel from a specified line and column number with respect to the current size 80x24 or 132x27
vtGetValueFor	Handy wrapper - get the input value (the east value) for the label
vtGetXlateDesc	Xlate descriptor from the virtual terminal to the client. Xlate descriptor from the client to the virtual terminal.
vtHex	Utility: Convert to readable hex.
vtHexChar	Utility: Convert to readable hex.
vtIsIceCap	??
vtJoblog	Text to place in the joblog.
vtLinCol	Convert a Lin and Coloum to Location structure.
vtLocate	Find the first occurrence of an input/output field relative to the search starting. If the search argument was not found, it returns *LOVAL (both line and col set to zero).
vtNow	?
vtOpen	Returns a pointer to a new Virtual Terminal Session (Keep that pointer for succeeding API calls).

Virtual Terminal API	Description
vtParseTrace	Post mortem, debug via a trace file; only supply the 5250 stream.
vtPressFormKey	Make a roundtrip to the session with input fields, your key pressed, and get to the next panel. The key is converted from a text string into a key code.
vtPressKey	To ensure that client program sets data in the emulator with the same codepage. Net needed if client codepage is the same as emulator session codepage. Make a roundtrip to the session with input fields, your key pressed, and get the next panel.
vtPrintf	Text to place in the trace
vtReloadOutputBuffer	Reload the output buffer (ScnBuf in the vts structutre) i.e. for vtGetOutputList()
vtSaaTitle	Returns the title of an SAA-compliant panel: 1) The title is assumed to be centered in the first line 2) It must be surrounded by highlight / white attributes
vtSaaTitleContains	Returns true / *ON if the value argument was found in the panel Title, which in turn must comply with the SAA standard: 1) The title is assumed to be centered. 2) It must be surrounded by highlight / white attributes.
vtSetCcsid	?
vtSetCursor	Set the cursor to a direct line/col.
vtSetCursorLoc	Set the cursor to a direct line/col via locations data structure.
vtSetCursorFieldNo	Set the cursor to the position of a field number.
vtSetField	Locates input from line/column and updates it if it exists.
vtSetFieldLoc	LLocates input from line/column and updates it if it exists.
vtSetFieldNo	Locates an input field and updates its value.
vtSetFieldNo2	Locates an input field and updates its value.
vtSetMetaData	Locates a field from a line/column data structure and updates it if it exists. The metadata is typically a JSON object that the Emulator can intercept.

Virtual Terminal API	Description
vtSetMetaDataLoc	Locates a field from a line/column data structure and updates it if it exists. The metadata is typically a JSON object that the Emulator can intercept.
vtTrim	Utility: Trim leading unprintable, return offset to first nonblank.
vtWinGetFieldNo	Locates an input field in a window and returns its current value. The field value, attributes, type, formatting, etc., are returned in the vtFieldDS.
vtWinGetLine	Returns a given line number from the current window. If no window is active, the line from the current display is returned. The line is stripped for display attributes.
vtWinGetLineRaw	Returns a given line number from the current window. If no window is active, the line from the current display is returned. The line is returned "as is" with display attributes and nonprintable characters.
vtWinGetScreen	Returns a substring of the current panel from a specified line and column with respect to the current size and location of a window. If no window is active, it will return the same value as vtGetScreen.
vtWinPatch	Patch a given string into the screen buffer in the relative window in the final buffer after decoration. It can also be used to remove data by inserting blanks.
vtWinPatchRaw	Patch a given string into the screen buffer in the relative window in the raw 5250 stream before any decoration. It can also be used to remove data by inserting blanks.
vtWinSetCursor	Set the cursor relative to the current window or fullscreen panel.
vtWinSetCursorLoc	Set the cursor relative to the current window or fullscreen panel.
vtWinTitle	Returns the title of an SAA-compliant window if the window is active. Otherwise, the title of the panel: 1) The title is assumed to be centered at the first line in the window. 2) It must be surrounded by highlight / white attributes.
vtWinTitleContains	Returns true / *ON if the value argument was found in the window Title, which in turn must comply with the SAA-standard: 1) The title is assumed to be centered. 2) It must be surrounded by highlight / white attributes.

Appendix E - Web Components ('xtype')

The front end of IceCap and Portfolio is developed using the ExtJS framework from Sencha. If you plan to develop your own components or use ExtJS for other purposes, please be aware that it is a licensed product. For more information, visit the [Sencha ExtJS homepage](#).

In ExtJS, these web components are referred to as 'xtype' and provide a shorthand configuration property to specify the type of a component. 'xtype' stands for "Extended Type" and enables developers to define UI components concisely within the framework.

IceCap includes components specifically designed to address common requirements when modernizing a 5250 interface. Many of these components are utilized in our demo environment and can be found in the application "Tags" within "IceCap Settings".

xtype	Description	Example in JavaScript
	These general keywords can be applied to every xtype: Mandatory: 'type' identify the component. 'width' is the width in pixels. 'flex' is equivalent to one character in a 5250 interface. Optional: 'tooltip' also known as MouseOver, it displays a message when the cursor hovers over the field. 'emptyText' also known as Placeholder, it provides an example or hint inside the field for users to see	<pre>{ "type": "" "width": 300, "flex": 30, "tooltip": "Field should be filled", "emptyText": "Please enter value" }</pre>
textfield	Text Field This can be used to adjust the size and provide relevant information for any text field.	<pre>{ "type": "textfield" "width": 500, "flex": 50, "tooltip": "This is a text field", "emptyText": "Enter text here" }</pre>
datefield	Calendar Shows the date from the field in a calendar. Date format: "d-m-y" = DD-MM-YY "d-m-Y" = DD-MM-YYYY	<pre>{ "type": "datefield", "format": "d-m-Y" }</pre>

xtype	Description	Example in JavaScript
	"j-m-Y" = D-MM-YYYY "m-d-Y" = MM-DD-YYYY "n-d-Y" = M-DD-YYYY	
numberfield	Number Field Right-aligned numeric value. The default decimal precision is 2. Decimals are displayed only when "allowDecimal" is set to true.	<pre>{ "type": "numberfield", "width": 300, "decimalPrecision": 4, "allowDecimal": true, "flex": 13 }</pre>
functionfield	Function Field Add a magnifying glass icon to the field as a tool to launch a program. The 'func' attribute defines the command or program to be called, with the field value potentially included in the call string. The 'initFunc' attribute specifies a command or program to execute prior to launching the main program."	<pre>{ "type": "functionfield", "func": "CALL PGM(CORPPGM/EMPLOYRC) PARM('B' '{arg}')", "initFunc": "CALL PGM(CORPPGM/SETLIBLC)", "flex": 8, "fixedSize": true }</pre>
actionfield	Action Field Add a magnifying glass icon to the field as a tool to trigger a function key, such as F4, as demonstrated in the example.	<pre>{ "type": "actionfield", "command": "F4" }</pre>
mapfield	Map Display the location on a map based on text input containing the country, city, street, etc. The service is provided by OpenStreetMap.	<pre>{ "type": "mapfield" }</pre>
websearch	Web Search Perform an internet search using the field content. Service provided by Bing.	<pre>{ "type": "websearch", "searchUrl": "https://www.bing.com/search?q={search}" }</pre>
checkboxfield	CheckBox A checkbox where 'X' returns true ('Y') and a blank returns false ('N').	<pre>{ "type": "checkboxfield", "trueValue": "Y", "falseValue": "N" }</pre>
radiogroup	Radio Buttons	<pre>{ "type": "radiogroup", "items": [</pre>

xtype	Description	Example in JavaScript
	Selects and returns only one input value ('S', 'M', or 'L') in the field labeled 'name'.	<pre> { "boxLabel": "Small", "name": "size", "inputValue": "S" }, { "boxLabel": "Medium", "name": "size", "inputValue": "M" }, { "boxLabel": "Large", "name": "size", "inputValue": "L" }] } </pre>
combobox	Combobox (Fixed values) The combobox displays three fixed text options, and when one is selected, the corresponding value is returned in the field.	<pre> { "type": "combobox", "flex": 16, "fieldStyle": {}, "values": [{ "text": "High School", "value": "1" }, { "text": "College", "value": "2" }, { "text": "University", "value": "3" }] } </pre>
DDSQL	DropDownSQL (Basic) Search by department number or name, or press [Cursor Down] to select from a complete list of departments. The 'valueColumn' returns the Department Number	<pre> { "type": "DDSQL", "library": "CORPDATA", "fileName": "DEPARTMENT", "flex": 30, "width": 400, "textColumns": ["DEPTNO", "DEPTNAME"], "valueColumn": "DEPTNO" } </pre>
DDSQL	DropDownSQL (Complex)	<pre> { "type": "DDSQL", "library": "*LIBL", </pre>

xtype	Description	Example in JavaScript
	Search for libraries by typing the first letters of the name, or press [Cursor Down] to select from a complete list of libraries. 'asRaw' allows input that is not in the list. 'hideTrigger' hides the tool that shows the list. 'popup' displays a small arrow in the corner of the field to indicate a dropdown, instead of a dedicated trigger. 'splitSearch' enables word-by-word search instead of searching the entire phrase. 'displayList' will in this example only display the selected value, not the combined text.	<pre> "fileName": "table (QSYS2.OBJECT_STATISTICS('*ALL', '*LIB'))", "flex": 10, "width": 60, "popup": true, "hideTrigger": true, "splitSearch": false, "asRaw": true, "displayList": ["value"], "textColumns": ["objname", "objtext"], "valueColumn": "objname" } </pre>
DDSQL	DropDownSQL (Advanced SQL) Provides a list of employee job titles to choose from, along with the frequency of each title. 'asRaw' allows the entry of new job titles that are not on the list."	<pre> { "type": "DDSQL", "library": "CORPDATA", "fileName": "EMPLOYEE", "flex": 15, "hideTrigger": true, "splitSearch": false, "asRaw": true, "displayList": ["value"], "textColumns": ["JOB concat ' (' CONCAT COUNT(JOB) CONCAT ')'"], "valueColumn": "JOB", "where": "JOB <> ' ' GROUP BY JOB ORDER BY JOB" } </pre>

Appendix F - Customized Menu-Loader

This appendix provides a step-by-step guide on creating a Menu-Loader using RPG Free and SQL, which generates a JSON response for the Web Client. For more detailed programming guidance, refer to the *IceBreak Programmer's Guide*.

Building a Menu Structure

The following example outlines how to construct a menu using RPG Free and SQL, which can be returned as a JSON response to a Web Client.

Note: The latest version of the Menu-Loader program can be found in the directory:

`\www\iceapp\icecap2\plugins\.`

Program Breakdown

Purpose

The Menu-Loader program is designed to load a system menu structure from a database and present it as a tree structure in JSON format for web navigation. The menu data is retrieved from the `menuloader` table and supports the navigation of various applications within the system.

Key Components

- The menu is built from the `menuloader` table, which includes fields such as `menu_id`, `parent_id`, `title`, and `link`. These fields define the hierarchy and navigation within the system.
- The program processes this structure into a JSON format and sends it as a JSON response (`application/json`) to the web client.

Main Functional Steps

- **Request Parsing:** The program inspects the incoming request for parameters (such as `payload`) or parses `POST` data to determine the starting point of the menu (e.g., `root` or a specified node).

- **SQL Query Execution:** It dynamically creates SQL queries to retrieve the relevant menu items and determines if nodes have child items. Leaf nodes (final menu items) and branch nodes (with further sub-items) are identified.
- **JSON Construction:** The program uses JSON routines and pointers to construct a JSON array representing the menu items, complete with their links and labels.
- **Permission Checking:** A function is implemented to verify if a user has permission to view certain menu items, ensuring secure access.

Main Procedure

This main procedure is responsible for:

- Loading the menu structure based on a specified node (or root node by default).
- Returning the structure as a JSON response to the client.

```
<%
ctl-opt copyright('System & Method (C), 2024');
ctl-opt decEdit('0,') datEdit(*YMD.) main(main);
ctl-opt text('Portfolio: System Menu loader');
ctl-opt bndDir('SNAPONISVR');
* -----
* Author .....: System & Method - November 2024
* HOME: CRTICEPGM STMF('/prj/icecap/plugins/menuloader.rpgle') SVRID([portsvr])
LIBL(*SERVER) OBJLIB([portsvr])
* AWAY: CRTICEPGM STMF('/www/iceapp/[icecap server]/plugins/menuloader.rpgle')
SVRID([portsvr]) LIBL(*SERVER) OBJLIB([portsvr])
* -----

/include qasphdr,iceUtility
/include qasphdr,jsonparser
/include qasphdr,snapOniSvr

/* -----
Main line:
----- */
dcl-proc main;

    dcl-s  pOutput          pointer;
    dcl-s  pRow             pointer;
    dcl-s  pRows           pointer;
    dcl-s  pPayload         pointer;
    dcl-s  sqlStmt          varchar(8192);
    dcl-s  node             varchar(32);
    dcl-s  link             varchar(8192);
    dcl-s  leaf            int(10);
```

```

dcl-ds rows                                likeds(json_iterator);

setContentType('application/json; charset=utf-8');
setEncodingType('*JSON');

pOutput = json_newArray();

if reqStr('payload') > '';
    pPayload = json_parseString(reqStr('payload'));
elseif getServerVar('REQUEST_METHOD') = 'POST';
    pPayload = json_parseRequest();
else;
    pPayload = *NULL;
endif;

node = json_getstr(pPayload : 'node' : 'root');

sqlStmt = (`
select    menu_id as menu_id
        , parent_id as parent_id
        , title as title
        , link as link
        , icecapinitfunc as initfunc
        , icecapmode as mode
        , case when
            (
                select count(menu_id)
                from menuloader cm
                where cm.parent_id = pm.menu_id
            ) <> 0 then 0 else 1 end as leaf
from menuloader as pm
`);

if node <> 'root';
    sqlStmt += (`
        where pm.parent_id = ${node}
    `);
else;
    sqlStmt += (`
        where pm.parent_id is null or pm.parent_id = 0
    `);
endif;

pRows = json_sqlResultSet(sqlStmt :1 :JSON_ALLROWS);
rows = json_setIterator(pRows);

dow json_forEach(rows);

    leaf = json_getInt(rows.this : 'LEAF': 0);

```

```

    if leaf = 1;

        link = lowercase(json_getstr(rows.this : 'LINK': ''));

        if      %scan('http': link) = 1
            or %scan('ftp': link) = 1
            or %scan('file': link) = 1;
            pRow = simpleLink(rows.this);
        else;
            pRow = icecapLink(rows.this);
        endif;

        else;
            pRow = simpleNode(rows.this);
        endif;

        json_arrayPush(
            pOutput
            :pRow
        );
    enddo;
    json_delete(pRows);

    responseWriteJson(pOutput);

    json_delete(pOutput);

    return;

end-proc;

```

icecapLink Procedure

This procedure creates a specialized link for IceCap applications. It builds a JSON object representing menu items with properties such as:

- **appName**: Application name derived from the menu title.
- **appFolder**: Path within IceCap.
- **initFunc**: Initial function to call within the IceCap view.
- **mode**: A specific configuration for the link (if defined). It also uses an icon fetching function to assign the appropriate icon for the menu item.

```
/* ----- */
```

```

Build icecap link
/* ----- */
dcl-proc icecapLink;

    dcl-pi *n                pointer;
           pInput            pointer value;
    end-pi;

    dcl-s pOutput            pointer;
    dcl-s pProperties         pointer;
    dcl-s pRequires          pointer;
    dcl-s pIcon              pointer;
    dcl-s title              varchar(512);
    dcl-s initFunc           varchar(256);
    dcl-s mode               varchar(100);

    if (not hasMenuOptionPermission(pInput));
        return *NULL;
    endif;

    title = json_getstr(pInput : 'TITLE');
    initFunc = json_getstr(pInput : 'INITFUNC');
    mode = json_getstr(pInput : 'MODE');

    pOutput = json_newObject();

    json_setstr(pOutput : 'text' : title);
    json_setstr(pOutput : 'title' : title);
    json_setbool(pOutput : 'leaf' : *ON);

    pProperties = json_newObject();

    json_setint(pProperties : '__cid__' : 4);
    json_setstr(
        pProperties
        : 'appName'
        : %scanRpl(' ' : '.' : lowercase(title))
    );

    json_setstr(pProperties : 'appFolder' : '/icecap/app');
    json_setbool(pProperties : 'advance' : *ON);
    json_setstr(pProperties : 'main' : 'IceCap.view.Main');
    json_setbool(pProperties : 'binding.alt' : *OFF);
    json_setbool(pProperties : 'binding.ctrl' : *OFF);
    json_setbool(pProperties : 'binding.shift' : *OFF);
    json_setnull(pProperties : 'binding.keyCode');
    json_setstr(pProperties : 'title' : title);

    pRequires = json_newArray();
    json_arrayPush(pRequires : 'IceCap.view.Main');

```

```
    json_moveObjectInto(
        pProperties
        : 'requires'
        : pRequires
    );

    json_setstr(pProperties : 'packages.IceCap': '/icecap/app');
    json_copyValue(
        pProperties : 'appConfig.func'
        : pInput : 'LINK'
    );

    // Init Func
    if initFunc <> '';
        json_setstr(
            pProperties
            : 'appConfig.initFunc'
            : initFunc);
    endif;

    // Mode
    if mode <> '';
        json_setstr(
            pProperties
            : 'appConfig.config'
            : mode);
    endif;

    pIcon = getIcon(title);
    json_copyValue(
        pProperties : 'icon'
        : pIcon : 'icon'
    );
    json_delete(pIcon);

    json_setstr(pProperties : 'seourl' : '');

    json_moveObjectInto(
        pOutput
        : 'properties'
        : pProperties
    );

    return pOutput;

end-proc;
```

simpleLink Procedure

Handles the creation of basic HTTP or external links. It retrieves basic properties like the menu item title and URL and creates a JSON object. The links can be HTTP, FTP, or file-based external links.

```

/* ----- */
  Build simple link (http... or GO)
/* ----- */
dcl-proc simpleLink;

  dcl-pi *n          pointer;
        pInput      pointer value;
  end-pi;

  dcl-s pOutput      pointer;
  dcl-s pProperties   pointer;
  dcl-s title        varchar(512);

  title = json_getstr(pInput : 'TITLE');

  pOutput = json_newObject();

  json_setstr(pOutput : 'text' : title);
  json_setstr(pOutput : 'title' : title);
  json_copyValue(pOutput : 'url' : pInput : 'LINK');
  json_setbool(pOutput : 'leaf' : *ON);

  // __cid__ == collection id / node type id
  // 2 group
  // 4 application

  pProperties = json_newObject();
  json_setint(pProperties : '__cid__' : 4);
  json_setbool(pProperties : 'binding.alt' : *OFF);
  json_setbool(pProperties : 'binding.ctrl' : *OFF);
  json_setbool(pProperties : 'binding.shift' : *OFF);
  json_setnull(pProperties : 'binding.keyCode');
  json_setstr(pProperties : 'icon' : '');
  json_setstr(pProperties : 'title' : title);
  json_copyValue(pProperties : 'url' : pInput : 'LINK');

  json_moveObjectInto(
    pOutput
    : 'properties'
    : pProperties
  );

  return pOutput;

```

```
end-proc;
```

simpleNode Procedure

This procedure builds nodes representing folders in the menu structure, which are non-leaf items. It constructs a JSON object representing the node, setting properties such as ID, title, and URL. These nodes often have child menu items.

```
/* ----- */
  Build simple Node
/* ----- */
dcl-proc simpleNode;

  dcl-pi *n          pointer;
        pInput      pointer value;
  end-pi;

  dcl-s pOutput      pointer;
  dcl-s pProperties   pointer;
  dcl-s title        varchar(512);
  dcl-s link         varchar(512);

  title = json_getstr(pInput : 'TITLE': '');
  link = json_getstr(pInput : 'link': '/menuLoader.aspx');

  pOutput = json_newObject();

  json_setInt(
    pOutput
    : 'id'
    : json_getInt(pInput: 'MENU_ID')
  );

  json_setstr(pOutput : 'text' : title);
  json_setstr(pOutput : 'title' : title);
  json_setbool(pOutput : 'leaf' : *OFF);
  json_setbool(pOutput : 'collapsed' : *ON);
  json_setStr(pOutput : 'url' : link);

  // __cid__ == collection id / node type id
  // 2 group
  // 4 application

  pProperties = json_newObject();
  json_setint(pProperties : '__cid__': 2);
```

```

json_setbool(pProperties : 'binding.alt' : *OFF);
json_setbool(pProperties : 'binding.ctrl' : *OFF);
json_setbool(pProperties : 'binding.shift' : *OFF);
json_setnull(pProperties : 'binding.keyCode');
json_setstr(pProperties : 'icon' : '');
json_setstr(pProperties : 'title' : title);
json_setStr(pProperties : 'url' : link);

json_moveObjectInto(
    pOutput
    : 'properties'
    : pProperties
);

return pOutput;

end-proc;

```

hasMenuOptionPermission Procedure

Checks whether the user has the permission to access a specific menu item. It retrieves the current user ID and determines whether they have the necessary permissions, always returning **true(*ON)** in this example.

```

/* ----- */
Check menu option permission
/* ----- */
dcl-proc hasMenuOptionPermission;

    dcl-pi *n            ind;
            pInput      pointer value;
    end-pi;

    dcl-s pRow           pointer;
    dcl-s sqlStmt        varchar(8192);
    dcl-s userId         varchar(128);
    dcl-s permission     ind;

    permission = *ON;

    userId = sesgetvar('ip2.navigator.user.id');

    return permission;

end-proc;

```

getIcon Procedure

This procedure determines the most appropriate icon for a menu item using the `findIcon()` function from the 'SNAPONISVR' binding directory. The icon is chosen based on keywords from the application title, using a scoring system to match the most relevant icon. The system supports both English and Danish, and it selects icons based on common terms used in ERP systems and the IBM i OS

```
/* ----- */
  Get icon
/* ----- */
dcl-proc getIcon;
  dcl-pi *n                pointer;
          txt              varchar(256) value;
  end-pi;

  dcl-s pOutput            pointer;
  dcl-s pParams            pointer;
  dcl-s pResult            pointer;

  pParams = json_newObject();
  json_setstr(
    pParams
    : 'keyword'
    : txt
  );
  pResult = findIcon(pParams);
  json_delete(pParams);

  pOutput = json_newObject();
  if (pResult <> *NULL);
    json_copyValue(
      pOutput : 'icon'
      : pResult : 'iconCls'
    );
  endif;
  json_delete(pResult);

  return pOutput;
end-proc;
```

Refer to the “Setup Menu-Loader” section in the *Menu Setup* chapter for more implementation details.

The Result

JSON Response Example

By running the Menu-Loader in a browser, the generated JSON response provides a structured list of menu items with various properties like `id`, `text`, `url`, and whether the item is a leaf (final link) or collapsed node.

Root Node

Example JSON for Root node: `http://MyIBMi:7700/menuLoader.aspx?payload={"node":"root"}`

```
[
  {
    "id": 1,
    "text": "Links on the internet",
    "title": "Links on the internet",
    "leaf": false,
    "collapsed": true,
    "url": "/menuLoader.aspx",
    "properties": {
      "__cid__": 2,
      "binding": {
        "alt": false,
        "ctrl": false,
        "shift": false,
        "keyCode": null
      },
      "icon": "",
      "title": "Links on the internet",
      "url": "/menuLoader.aspx"
    }
  },
  {
    "id": 5,
    "text": "IBMi Commands",
    "title": "IBMi Commands",
    "leaf": false,
    "collapsed": true,
    "url": "/menuLoader.aspx",
    "properties": {
      "__cid__": 2,
      "binding": {
        "alt": false,
        "ctrl": false,
        "shift": false,
        "keyCode": null
      },
    },
  },
]
```

```
        "icon": "",
        "title": "IBMi Commands",
        "url": "/menuLoader.aspx"
    }
}
]
```

Content of Node #5

To view the next level of the menu, execute the Menu-Loader with a specific node number to display its content:

Example JSON for a Node: [http://myIBMi:7700/menuLoader.aspx?payload={"node":5}](http://myIBMi:7700/menuLoader.aspx?payload={)

```
[
  {
    "text": "Work spool file",
    "title": "Work spool file",
    "leaf": true,
    "properties": {
      "__cid__": 4,
      "appName": "work.spool.file",
      "appFolder": "/icecap/app",
      "advance": true,
      "main": "IceCap.view.Main",
      "binding": {
        "alt": false,
        "ctrl": false,
        "shift": false,
        "keyCode": null
      },
      "title": "Work spool file",
      "requires": [
        "IceCap.view.Main"
      ],
      "packages": {
        "IceCap": "/icecap/app"
      },
      "appConfig": {
        "func": "wrksplf",
        "config": "flow"
      },
      "icon": "pf-data",
      "seourl": ""
    }
  },
  {
```

```
"text": "Work system status",
"title": "Work system status",
"leaf": true,
"properties": {
  "__cid__": 4,
  "appName": "work.system.status",
  "appFolder": "/icecap/app",
  "advance": true,
  "main": "IceCap.view.Main",
  "binding": {
    "alt": false,
    "ctrl": false,
    "shift": false,
    "keyCode": null
  },
  "title": "Work system status",
  "requires": [
    "IceCap.view.Main"
  ],
  "packages": {
    "IceCap": "/icecap/app"
  },
  "appConfig": {
    "func": "wrksyssts"
  },
  "icon": "pf-server",
  "seourl": ""
}
},
{
  "text": "Command entry",
  "title": "Command entry",
  "leaf": true,
  "properties": {
    "__cid__": 4,
    "appName": "command.entry",
    "appFolder": "/icecap/app",
    "advance": true,
    "main": "IceCap.view.Main",
    "binding": {
      "alt": false,
      "ctrl": false,
      "shift": false,
      "keyCode": null
    },
    "title": "Command entry",
    "requires": [
      "IceCap.view.Main"
    ],
    "packages": {
```

```
        "IceCap": "/icecap/app"
      },
      "appConfig": {
        "func": "call qcmd"
      },
      "icon": "pf-megaphone",
      "seourl": ""
    }
  }
]
```

Understanding the JSON Structure

The JSON structure serves to configure and display menu items by defining both basic and advanced properties. It provides essential information such as display text, title, and functional properties like URLs, key bindings, and whether the item contains sub-items. In addition, it specifies advanced configurations depending on the menu item type, such as application location, main view, dependencies, keybinding options, and icon representation.

JSON Labels and Their Descriptions

"id"	A unique identifier for the menu item. Each menu item must have a distinct <code>id</code> value to ensure proper referencing.
"text"	The display text shown in the menu for the user to click. It provides a brief description or name of the menu item.
"title"	The title offers more information about the menu item, often used for tooltips or additional display purposes, and usually repeats the value in <code>text</code> .
"leaf"	Indicates whether the menu item is a final node (<code>true</code>) or if it contains sub-items (<code>false</code>).
"collapsed"	Specifies whether the menu item should appear collapsed (<code>true</code>), hiding sub-items, or expanded (<code>false</code>), showing sub-items by default.

"url"	Defines the URL the menu item links to. When clicked, the application will navigate to or load content from this URL. Example: <code>"/menuLoader.aspx"</code> .
"properties": { ... }	This section groups additional metadata or configurations related to the menu item, such as behavior, appearance, and binding options.

Additional Labels in "properties"

"__cid_"	Identifies the menu item type: 1 = Root?, 2 = Folder, 3 = Link?, 4 = IceCap Application, 5 = Model Studio Application?
"binding": { ... }	Defines key binding properties, allowing the menu item to be activated with keyboard shortcuts. Fields like <code>alt</code> , <code>ctrl</code> , <code>shift</code> , and <code>keyCode</code> define the trigger. or a specific key to activate this menu item.
"icon"	Defines the icon associated with the menu item, often used to visually represent the action or functionality. If left blank, a default icon for the menu item type will be shown. Example: <code>"pf-megaphone"</code>
"title"	This repeats the title for consistency within the <code>properties</code> section. It may be used in different contexts or for different elements of the interface, like tooltips.
"url":	Duplicates the <code>url</code> value, specifying where the menu item will navigate. Example: <code>"/menuLoader.aspx"</code> .

Additional Labels in "properties" for IceCap Applications:

"appName"	A unique name of the application or command being executed, used by Portfolio to check if the application is already active. By default already active applications will be opened instead of launched as new. Example: <code>"command.entry"</code> .
------------------	--

"appFolder"	Indicates the directory where the application is located. The value <code>"/icecap/app"</code> is mandatory for IceCap Applications.
"advance"	A Boolean flag (<code>true</code> or <code>false</code>) indicating if the application requires advanced properties to launch.
"main"	Specifies the main component of the application to be loaded when the menu item is clicked. Example: <code>"IceCap.view.Main"</code> .
"requires"	Lists dependencies needed for this menu item to function. Example: <code>"IceCap.view.Main"</code> .
"packages": { ... }	Defines additional packages required for the menu item, specifying their locations. Example: <code>"IceCap": "/icecap/app"</code> .
"appConfig": { ... }	This section contains specific configuration details for the application: <code>"func": "call qcmd"</code>: Specifies the command or function that is executed <code>"config": "flow"</code>: Specifies the presentation modes, like "Flat", "Flex", "Flow", and "Flip". If left blank "Flat" is chosen as default.
"seourl": ""	Used for SEO (Search Engine Optimization) or web-based access. If left blank, no specific SEO URL is provided. What is the difference between "seourl" and "url"?

Difference Between **"seourl"** and **"url"**

- "url"**: Specifies the actual URL that the menu item links to, determining where the application will navigate when selected. Example: `"/menuLoader.aspx"`.
- "seourl"**: Primarily used for SEO purposes, this label defines a search engine-friendly version of the URL. If left blank, no SEO-specific URL is provided.

Appendix V - Operations and System Monitoring

Managing the Solution Stack

The solution stack contain the following core products, each serving a role in modernizing and web-enabling IBM i applications.

IceBreak

IceBreak is a robust server framework designed to web-enable IBM i applications by acting as middleware between the IBM i system and web technologies. It serves as the backbone in IceCap and Portfolio for transforming 5250 programs into modern, browser-accessible applications without altering their core logic. By supporting multiple web protocols, IceBreak ensures seamless integration, secure web services, and dynamic session management. This makes legacy applications more efficient and accessible through any browser, enhancing both functionality and user experience.

Location	Library: ICEBREAK IFS path: /www/iceapp/IceBreak
Start	Command: ICEBREAK/STRICESBS Alternative: STRSBS ICEBREAK
Adminstration	Command: GO ICEBREAK Web Browser: http://myIBMi:7000
Stop	Command: ICEBREAK/ENDICESBS Alternative: ENDSBS ICEBREAK OPTION(*IMMED
Version	Command: DSPDATARA ICEBREAK/SVCVER

Portfolio

Portfolio is a web-based platform that transforms traditional IBM i 5250 environments into intuitive, multitasking systems. It organizes applications into a tree structure and supports roles and permissions for users, allowing efficient navigation and simultaneous access to multiple applications. This tab-based system enhances productivity and integrates seamlessly with web and cloud-based applications, providing a modernized, user-friendly experience.

Location	Library: ICEMNU, ICEMNUDB IFS path: /www/iceapp/Portfoilo
Start	Command: ICEBREAK/STRICESVR SVRID(ICEMNU)
Adminstration	Web Browser: http://myIBMi:7700
Stop	Command: ICEBREAK/ENDICESVR SVRID(ICEMNU)
Version	Command: DSPDATARA ICEMNU/PORTFOLIO

IceCap

IceCap is a web-enablement solution that transforms IBM i 5250 programs into modern, browser-based interfaces. It introduces features such as multitasking, advanced data validation, and modern data entry tools like checkboxes and date pickers. IceCap enables flexible workflows by seamlessly integrating with third-party systems, without changing the core program logic. Multiple presentation modes are available, enabling businesses to tailor the interface to their specific needs.

Location	Library: ICECAP IFS path: /www/iceapp/IceCap
Start	Command: ICEBREAK/STRICESVR SVRID(ICECAP)
Terminal	Web Browser: http://myIBMi:5250
Stop	Command: ICEBREAK/ENDICESVR SVRID(ICECAP)
Version	Command: DSPDATARA ICECAP/ICECAP

Monitoring and System Management

Efficient operation and system performance depend heavily on monitoring the IceBreak and QINTER subsystems. Ensuring that both are functioning optimally is critical for maintaining a stable environment.

Monitoring the IceBreak Subsystem

When the IceBreak subsystem starts, servers for IceBreak Administration, Portfolio, and optionally IceCap, are initiated as jobs within the subsystem. Each of these servers is managed by two balancing jobs:

1. **ICEMNU** with status SELW (Selection Wait)
2. **ICEMNU** with status LCKW (Lock Wait)

The job in **SELW** status initiates new sessions, while the job in **LCKW** status acts as a backup, ready to take over if the first job encounters a failure. For every login session in Portfolio, a new job is created. The logged-in user's name replaces the system default, allowing the system to track the ownership of each session.

Job Naming

Job names consist of the first six characters identifying the originating server, followed by a unique identifier of up to three characters (e.g., **ICEMNU_AB**, **ICEMNU_AC**).

Server Type

All servers, including IceBreak Administration, Portfolio, and IceCap, should be configured to run as ***MULTITHREAD**. This multithreading setup allows the server to efficiently handle multiple user sessions simultaneously by assigning separate threads for various tasks, such as HTTP requests, database interactions, and form processing. Multithreading ensures high performance under heavy

traffic and maintains session persistence, which is critical for running 5250 applications in a web environment.

You can optionally configure session persistence to restore all active tabs when Portfolio or the browser is reopened after a forced close.

For further configuration details, refer to the IceBreak manual.

Monitoring the QINTER Subsystem

Each time a user opens a tab or window in Portfolio, a virtual workstation device prefixed by "ICE" is created within the IBM i system. These virtual devices correspond to interactive jobs managing the user sessions. When a tab or window is closed, the 5250 job associated with that virtual device is ended, and the device is varied off but remains available for future sessions.

Relevant operations commands:

<code>WRKACTJOB SBS(QINTER) JOB(ICE*)</code>	Displays all active 5250 jobs related to IceCap, running as tabs or windows in Portfolio.
<code>WRKCFGSTS CFGTYPE(*DEV) CFGD(ICE*)</code>	Displays device descriptions that are in use or available for IceCap.

Securing Client/Server Communication

To prevent disruptions in communication between the client and server, IceCap implements robust features designed to maintain session integrity, especially given the sensitivity of the 5250 protocol to connection interruptions.

Keep-Alive Feature

Every 60 seconds, the web client sends a signal to the server to confirm the session's activity. This mechanism helps prevent premature termination of sessions due to inactivity or communication delays, ensuring a stable and continuous user experience.

Monitor Job

A dedicated monitor job runs on the server, overseeing all 5250 jobs initiated by IceCap. This monitor checks the timestamp of the last keep-alive signal. If a job has not been active for more than 60 seconds, the monitor will perform a controlled termination using the `ENDJOB OPTION(*CNTRLD)` command. If the session has not been gracefully terminated within an additional 60 seconds, the monitor forcibly ends the job with the `ENDJOB OPTION(*IMMED)` command. Once the session ends, the corresponding virtual device is varied off and made available for future use.

- **Monitor Identification:** The monitor job within the Portfolio subsystem is identified as `<PortfolioServer>_MON` (e.g., `ICEMNU_MON`).

5250 jobs may be abandoned if the user closes the browser tab or window and ignores warnings from Portfolio, or if their device enters sleep mode or shuts down unexpectedly. In such cases, the monitor terminates the job after exceeding the timeout period. Keeping both the keep-alive feature and the monitor job active ensures that these types of disruptions are minimized, maintaining system stability.

Appendix W - User Authentication and Data Integrity

IceCap and Portfolio follows robust security protocols to ensure user authentication, data integrity, and overall system security. These protocols are designed to comply with industry standards, making IceCap a secure solution for modernizing 5250 applications in web-based environments. Below is an outline of the key security measures:

User Authentication

IceCap adheres to the established user authentication mechanisms of the IBM i platform, ensuring that only authorized users can access the system. The security protocols include:

- **Integration with IBM i Authentication:** IceCap leverages IBM i's existing user authentication system, meaning it uses the same user profiles and access controls defined in IBM i.
 - **User Profiles:** Users access IceCap with the same credentials used for logging into IBM i, ensuring consistency and security.
 - **Role-based Access Control:** Access to applications and data is controlled through user roles and permissions. These roles are managed within IBM i and enforced in IceCap.
 - **Single Sign-On (SSO):** If configured, IceCap can support single sign-on, allowing users to authenticate once and gain access to multiple applications within the system without having to log in repeatedly.
 - **Session Management:** IceCap manages user sessions securely. When a user logs in, their session is authenticated, and IceCap maintains this session throughout their use of the web interface. Session timeouts and secure session handling ensure that sessions cannot be hijacked or misused.
-

Data Integrity

IceCap ensures that the data processed, displayed, and transmitted remains accurate and secure. Key measures to maintain data integrity include:

- **Encryption:** IceCap supports encrypted communication between the client and server to protect data in transit.
 - **HTTPS (SSL/TLS):** All web-based communication between IceCap and the user's browser is secured using HTTPS. This ensures that data exchanged during user sessions, such as form submissions and queries, is encrypted and cannot be intercepted by unauthorized parties.
 - **Database Integrity:** IceCap ensures that data from the underlying IBM i database is accurately represented in the web interface. Any updates or changes made by users in the web application are validated and securely transmitted back to the database without altering data integrity.
 - **Access Control:** IceCap enforces IBM i's access control mechanisms, ensuring that users only have access to the data and programs that their roles permit. Unauthorized users cannot access sensitive data or modify critical applications.
-

Security Compliance with IBM i

Since IceCap runs natively on IBM i, it inherits all the security protocols of the IBM i operating system. This includes:

- **Object-Level Security:** IceCap respects object-level security defined within IBM i. This means that users can only access files, programs, or data they are authorized to view or edit.
 - **Job and Process Monitoring:** IBM i's job control and process monitoring capabilities are leveraged to ensure that IceCap processes are secure and isolated from unauthorized access.
 - **Encryption and Data Masking:** IBM i supports encryption of data at rest, and IceCap follows these protocols to ensure sensitive data stored on the system is protected.
-

Session Security and Persistence

IceCap's session management ensures that:

- Sessions are secured and managed to prevent unauthorized access. If a session becomes inactive, it will be terminated after a set timeout to prevent misuse.

- Session **persistence** can be configured to restore a user's active session and tabs when they reopen IceCap, but this is optional and can be controlled depending on security policies.

Optional Security Features

- **Forced Logout and Session Expiry:** IceCap can be configured to automatically log users out after a period of inactivity, reducing the risk of unauthorized access due to forgotten open sessions.
- **Secure Cookies and CSRF Protection:** IceCap follows secure cookie practices and can support Cross-Site Request Forgery (CSRF) protection, reducing the risk of cross-site attacks.

Summary of Security Protocols

Security Aspect	Protocol
User Authentication	Integration with IBM i user profiles and authentication mechanisms, role-based access control.
Encryption	HTTPS (SSL/TLS) for securing data in transit.
Session Management	Secure session handling, session timeouts, optional session persistence.
Data Integrity	Enforced access controls, accurate data representation, database integrity.
IBM i Compliance	Inherits IBM i object-level security, job/process monitoring, and encryption capabilities.
Session Security	Session timeouts, forced logout after inactivity, secure cookies, and CSRF protection.

Appendix X - Troubleshooting and Technical Support

Reporting Technical Issues

For technical support, please contact our support team at support@system-method.com. To expedite and enhance the support process, please provide the following information:

- Full screenshot of the 5250 screen in question.
- Full screenshot of the IceCap result, including the mode (Flat/Flex/Flow/Flip).
- The entire `icecap_config.xml` file.
- The source code for the emulator program.
- The source code for the interpretation and navigation programs, if applicable.

Tracing Issues in IceCap

To help us diagnose issues effectively, you can generate a trace file from IceCap, which allows us to replicate the 5250 sequence in our QA environment. Follow these steps to set up the trace:

1. Locate the `icecap_config.xml` file on the IFS.
2. Search for the trace configuration within the file.
3. Enable tracing by modifying your configuration as follows:

```
<corpflow>
  <defaults
    trace="no"
    tracePath="/www/iceapp/icecqa/trace/"
    logClientActions="no"

  />
</corpflow>
```

4. Set a trace path (e.g., `/www/iceapp/trace/`).
5. Create the trace path folder (e.g., `/www/iceapp/trace/`).
6. Set access permissions for the trace path folder using the command:

```
WRKAUT OBJ('/www/iceapp/trace')
```

Ensure the folder has public read, write, and execute permissions, along with full authority (*public *RWX + *ALL):

```

                                Work with Authority

Object . . . . . : /www/iceapp/icecqa/trace
Type . . . . . : DIR
Owner . . . . . : PNO
Primary group . . . . . : *NONE
Authorization list . . . . . : *NONE

Type options, press Enter.
  1=Add user   2=Change user authority   4=Remove user

Opt  User      Data Authority  --Object Authorities--
      User      Authority  Exist  Mgt  Alter  Ref
--
_    *PUBLIC    *RX      X      X    X    X
_    PNO        *RWX     X      X    X    X
_

```

- 7. Restart the server instance.
- 8. Execute the sequence of 5250 screens as cleanly as possible to avoid irrelevant screen data in the trace file.
- 9. Attach the generated trace file to your support email.

By following these steps, you will provide our support team with the necessary data to diagnose and resolve your issue more efficiently.

Appendix Y - License Agreement

This license agreement applies to the downloadable version of the software. By using, copying, transmitting, distributing, or installing the software, you agree to all of the terms of this License. If you do not agree to any of the terms of this License, then do not use, copy, transmit, distribute, or install the software.

You may, without making any payment to System & Method A/S:

1. Give exact copies of this software evaluation version personally to anyone.
2. Distribute exact copies of this version of the software.
3. Make as many exact copies of this evaluation version of the software as you wish for purposes of distribution as described in (a) and (b) above.

You are expressly prohibited from charging or requesting donations for any copies, however, made, and from distributing such copies with other products of any kind, commercial or otherwise, without prior written permission from System & Method A/S. System & Method A/S reserves the right to revoke the above distribution rights at any time for any or no reason.

This software and all accompanying files, data, and materials are distributed "AS IS" and with no warranties of any kind, whether express or implied. Good data processing procedure dictates that any program be thoroughly tested with non-critical data before relying on it.

The user must assume the entire risk of using the program. This disclaimer of warranty constitutes an essential part of the Agreement. In no event shall System & Method A/S, or its principals, shareholders, officers, employees, affiliates, contractors, subsidiaries, or parent organizations, be liable for any incidental, consequential, or punitive damages whatsoever relating to the use of the software, or your relationship with System & Method A/S.

In addition, in no event does System & Method A/S authorize you to use the software in applications or systems where the software failure to perform can reasonably be expected to result in a significant physical injury or loss of life. Any such use by you is entirely at your own risk, and you agree to hold System & Method A/S harmless from any claims or losses relating to such unauthorized use.

This Agreement is the complete statement of the Agreement between the parties on the subject matter and merges and supersedes all other or prior understandings, purchase orders, agreements, and arrangements. The laws of the Country of Denmark shall govern this Agreement. Exclusive

jurisdiction and venue for all matters relating to this Agreement shall be in courts and fora located in the country of Denmark, and you consent to such jurisdiction and venue.

All rights of any kind in the software, which are not expressly granted in this License, are entirely and exclusively reserved to and by System & Method A/S. You may not rent, lease, modify, translate, reverse engineer, decompile, disassemble, or create derivative works based on the software. You may not make access to the software available to others in connection with a hosting center, application service provider, or similar business or use the software in a business to provide file compression, decompression, or conversion services to others. There are no third-party beneficiaries of any promises, obligations, or representations made by System & Method A/S herein.

© Copyright by System & Method A/S. All rights reserved.

Appendix Z - RPG Source Repository

This appendix provides access to the RPG program sources related to the following areas and components:

- Configuration
- Emulation
- Interpretation
- Navigation
- Organization (PC Organizer)
- Virtual Terminal API (VT)

These sources serve as valuable resources for documentation, inspiration, and examples of best practices within the various components. They are intended to help you understand and customize the functionality as needed.

Please note that these sources do not represent the complete components; they provide methods to intervene at specific points within each component.

Recommended Tools

We recommend using [Visual Studio Code \(VS Code\)](#) by Microsoft as your code editor. For enhanced RPG editing capabilities, you can use our open-source plugin available at: [RPG for VSCode](#).

Important Notes

- **Backup:** Always create a copy of the original source files before making any modifications.
- **Compilation Setup:** Follow the compilation instructions provided in the comments at the top of the source files.
- **Version Differences:** Be aware that these sources are subject to updates and may differ from those included in your current release. The corresponding sources are distributed with each product release and can be found in the following directory:
[/www/iceapp/IceCap2/plugins.](#)

By following these guidelines, you can effectively utilize the RPG sources for your development and customization needs.

Configuration

The XML configuration file `icecap_config.xml` contains the detailed technical settings necessary for configuring IceCap. It is important to note that this is not a program source file, and therefore, it cannot be compiled. Any changes made to this XML file will only take effect after the IceCap server has been restarted.

```
<?xml version="1.0" encoding="UTF-8" ?>
<config>
  <default>    <!-- Default config. All Tags can be override -->

    <!-- For default general theme. Look in folder: /www/system/styles/ext/xtheme*
(extention .css) -->
    <!-- For default IceCap theme. Look in xml file: /www/icecap/icecap_theme.xml -->

    <defaults
      trace="yes"
      tracePath="/www/iceapp/icecqa/trace_flat/"
      logClientActions="no"
      contentLocation="east"
      timeOut="60"
      detectWindows="always"

      showTitle="yes"
      showOptionText="no"
      showOptionField="no"
      showOptionColumn="yes"
      showOptionNo="yes"
      paragraphs="="
      defaultOptions="2,1,5"
      alternativeOptions=""
      showAllKeys="no"
      showKeysText="no"
      showKeysAsButtons="yes"
      showKeysAsTabs="no"
      showKeysAsPredefined=""
      showKeysFxx="yes"
      showKeysToolTip="no"
      showKeySeparator="="
      showKeysMenu="no"
      showStatusBar="yes"
      showPageUpDown="no"
      showLabelUnderlined="no"
      showEnterKey="no"
      enterKeyText="OK"
      helpKey="F1"
      exitKey="F3"
      promptKey="F4"
```

```

        refreshKey="F5"
        cancelKey="F12"
        moreOptionsKey="F23"
        moreKeysKey="F24"
        messageline=""
        optionsSplitValue="  "
        columnHeaderAttribute= "22"
        useInterpretation="no"
        useTags="no"
        useSubfileTransformation="no"
        useScreenTranslation="no"
        translateToLanguages="da,en,de"
    />

    <emulator
        program="/menu/icecap.aspx"
        useHints="no"
        hintsXMLhive=""
        hintsXMLpath=""
        numberOfSubfilePagesToStack=""
    />

    <!-->
    <automaticScreenScripting>
        <!-- English -->
        <screen title="Sign Off" action="ENTER" />
        <screen title="Attempt to Recover Interactive Job" fieldNo="1" value="90"
action="ENTER" />
        <screen title="Display Messages" action="ENTER" />
        <screen title="Display Program Messages" action="ENTER" />
        <screen title="Sign-on Information" action="ENTER" />
        <!-- Danish -->
        <screen title="VIS PROGRAMMEDELELSER" action="ENTER" />
        <screen title="LOG PÅ" action="ENTER" />
        <!-- Finnish -->
        <screen title="Uloskirjaus" action="ENTER"/>
        <screen title="Sisäänkirjaus" action="ENTER"/>
        <screen title="Sisäänkirjaustiedot" action="ENTER"/>
        <screen title="Vuorovaikutteisen työn elvytysyritys" fieldNo="1" value="90"
action="ENTER"/>
        <screen title="Ohjelmanomien näyttö" action="ENTER"/>
        <!-- German -->
        <screen title="Abmelden" action="ENTER" />
        <screen title="Versuchen, interaktiven Job wiederherzustellen" fieldNo="1" value="90"
action="ENTER" />
        <screen title="Programmnachrichten anzeigen" action="ENTER" />
        <screen title="Anmeldeinformationen" action="ENTER" />
        <!-- Norwegian -->
        <screen title="Avslutte systemet" action="ENTER" />
        <screen title="Prøve å gjenopprette interaktiv jobb" fieldNo="1" value="90"

```

```

action="ENTER" />
    <screen title="Vise programmeldinger" action="ENTER" />
    <screen title="Opplysninger om påmelding" action="ENTER" />
    <!-- Sweden -->
    <screen title="Logga av" action="ENTER" />
    <screen title="FÖRSÖK ATT REKONSTRUERA INTERAKTIVT JOBB" fieldNo="1" value="90"
action="ENTER" />
    <screen title="Visa programmeddelanden" action="ENTER" />
    <screen title="Påloggningsinformation" action="ENTER" />
    <screen title="Visa meddelanden" action="ENTER" />
    <!-- Spanish -->
    <screen title="Finalizar sesión" action="ENTER"/>
    <screen title="Intentar Recuperar Trabajo Interactivo" action="ENTER" value="90"
fieldNo="1"/>
    <screen title="Visualizar mensajes" action="ENTER"/>
    <screen title="Visualizar Mensajes de Programa" action="ENTER"/>
    <screen title="Información de inicio de sesión" action="ENTER"/>
</automaticScreenScripting>

<breakOut
    navigatorProgram="*LIBL/*NONE"
    emulatorProgram="*LIBL/*NONE"
    pcoProgram="*LIBL/PCOSERVER"
    deviceNameProgram="*LIBL/*NONE"
    i18nXlateProgram="*LIBL/*NONE"
    varyoffExitProgram="*LIBL/*NONE"
/>
</default>

<corpflex>
    <defaults
        trace="no"
        tracePath="/www/iceapp/icecqa/trace/"
        logClientActions="no"

        showAllKeys="yes"
        showKeySeparator="- "
        showLabelUnderlined="yes"
        useInterpretation="yes"
        useTags="yes"
    />
    <breakOut
        navigatorProgram="*LIBL/*NONE"
        emulatorProgram="*LIBL/EMLFLXCORP"
        interpretationProgram="*LIBL/INTTAG"
        pcoProgram="*LIBL/PCOSERVER"
        i18nXlateProgram="*LIBL/*NONE"
        xxx_deviceNameProgram="*LIBL/*NONE"
        xxx_varyoffExitProgram="*LIBL/*NONE"
    />

```

```
</corpflex>

<corpflow>
  <defaults
    trace="no"
    tracePath="/www/iceapp/icecqa/trace/"
    logClientActions="no"

    showOptionSeparator="-"
    showKeysToolTip="yes"
    showKeySeparator=" "
    showKeysMenu="yes"
    showLabelUnderlined="yes"
    useInterpretation="yes"
    useTags="yes"
    useSubfileTransformation="yes"
  />
  <breakOut
    navigatorProgram="*LIBL/NAVCAPCORP"
    emulatorProgram="*LIBL/EMLFLWCORP"
    interpretationProgram="*LIBL/INTTAG"
    pcoProgram="*LIBL/PCOSERVER"
    i18nXlateProgram="*LIBL/*NONE"
    xxx_deviceNameProgram="*LIBL/*NONE"
    xxx_varyoffExitProgram="*LIBL/*NONE"
  />
</corpflow>

<corpflip>
  <defaults
    trace="no"
    tracePath="/www/iceapp/icecqa/trace/"
    logClientActions="no"

    showOptionColumn="no"
    showOptionNo="no"
    showKeysAsTabs="yes"
    showKeysAsPredefined="F6,F9"
    showKeysFxx="no"
    showKeysToolTip="yes"
    showStatusBar="no"
    showPageUpDown="yes"
    showLabelUnderlined="yes"
    showEnterKey="yes"
    enterKeyText="OK"
    useInterpretation="yes"
    useTags="yes"
    useSubfileTransformation="yes"
  />
  <breakOut
```

```
    navigatorProgram="*LIBL/NAVCAPCORP"
    emulatorProgram="*LIBL/EMLFLWCORP"
    interpretationProgram="*LIBL/INTTAG"
    pcoProgram="*LIBL/PCOSERVER"
    i18nXlateProgram="*LIBL/*NONE"
    xxx_deviceNameProgram="*LIBL/*NONE"
    xxx_varyoffExitProgram="*LIBL/*NONE"
  />
</corpflip>

<flex>
  <defaults
    trace="no"
    tracePath="/www/iceapp/icecqa/trace/"
    logClientActions="no"

    showAllKeys="yes"
    showKeySeparator="- "
    showLabelUnderlined="yes"
    useInterpretation="yes"
    useTags="yes"
  />
  <breakOut
    navigatorProgram="*LIBL/*NONE"
    emulatorProgram="*LIBL/EMLFLX"
    interpretationProgram="*LIBL/INTTAG"
    pcoProgram="*LIBL/PCOSERVER"
    i18nXlateProgram="*LIBL/*NONE"
    xxx_deviceNameProgram="*LIBL/*NONE"
    xxx_varyoffExitProgram="*LIBL/*NONE"
  />
</flex>

<flow>
  <defaults
    trace="yes"
    tracePath="/www/iceapp/icecqa/trace/"
    logClientActions="no"

    showOptionSeparator="- "
    showKeysToolTip="yes"
    showKeySeparator=" "
    showKeysMenu="yes"
    showLabelUnderlined="yes"
    useInterpretation="yes"
    useTags="yes"
    useSubfileTransformation="yes"
  />
  <breakOut
```

```

        navigatorProgram="*LIBL/*NONE"
        emulatorProgram="*LIBL/EMLFLW"
        interpretationProgram="*LIBL/INTTAG"
        pcoProgram="*LIBL/PCOSERVER"
        i18nXlateProgram="*LIBL/*NONE"
        __deviceNameProgram="*LIBL/JHODEVICE"
        xxx_varyoffExitProgram="*LIBL/*NONE"
    />
</flow>

<flip>
    <defaults
        trace="no"
        tracePath="/www/iceapp/icecqa/trace/"
        logClientActions="no"

        showOptionColumn="no"
        showOptionNo="no"
        showKeysAsTabs="yes"
        showKeysAsPredefined="F6,F9"
        showKeysFxx="no"
        showKeysToolTip="yes"
        showStatusBar="no"
        showPageUpDown="yes"
        showLabelUnderlined="yes"
        showEnterKey="yes"
        enterKeyText="OK"
        useInterpretation="yes"
        useTags="yes"
        useSubfileTransformation="yes"
    />
    <breakOut
        navigatorProgram="*LIBL/*NONE"
        emulatorProgram="*LIBL/EMLFLW"
        interpretationProgram="*LIBL/INTTAG"
        pcoProgram="*LIBL/PCOSERVER"
        i18nXlateProgram="*LIBL/*NONE"
        xxx_deviceNameProgram="*LIBL/*NONE"
        xxx_varyoffExitProgram="*LIBL/*NONE"
        clientXtypesPath="/menu/scripts/xtypes.js"
        clientScriptPath="/menu/scripts/breakouts.js"
        clientCSSPath="/menu/styles/changeFontSize.css"
    />
</flip>

<mobflow>
    <defaults
        optionsSplitValue=", "
        columnHeaderAttribute="22"
    />

```

```

<breakOut
  navigatorProgram="*LIBL/*NONE"
  emulatorProgram="*LIBL/MOBFLOW"
  interpretationProgram="*LIBL/INTTAG"
  pcoProgram="*LIBL/PCOSERVER"
  i18nXlateProgram="*LIBL/*NONE"
  xxx_deviceNameProgram="*LIBL/*NONE"
  xxx_varyoffExitProgram="*LIBL/*NONE"
/>
</mobflow>

  <!-- Do not delete!!!! -->
  <!-- Used in the IBMi menu application -->
  <!-- -->
<ibmimenu>
  <menutop>
    <key text="following:"/>          <!-- English -->
    <key text="følgende:"/>          <!-- Danish -->
    <key text="vaihtoehtoista:"/>    <!-- Finnish -->
    <key text="Auswahlmöglichkeiten:"/> <!-- German -->
    <key text="alternativ:"/>        <!-- Norwegian -->
    <key text="Alternativ:"/>        <!-- Sweden -->
    <key text="siguientes:"/>        <!-- Spanish -->
    <key text="opciones:"/>          <!-- Spanish -->
    <key text="opción:"/>            <!-- Spanish -->
  </menutop>
  <!-- Sweden -->
  <menubottom>
    <key text="Type choice or command"/> <!-- English -->
    <key text="Selection or command"/> <!-- English -->
    <key text="Indtast valg eller kommando"/> <!-- Danish -->
    <key text="Valinta tai komento"/> <!-- Finnish -->
    <key text="Auswahl oder Befehl"/> <!-- German -->
    <key text="Valg eller kommando"/> <!-- Norwegian -->
    <key text="Val eller kommando" /> <!-- Sweden -->
    <key text="Val eller kommando"/> <!-- Spanish -->
    <key text="Selección o mandato"/> <!-- Spanish -->
  </menubottom>
</ibmimenu>

</config>

```

Emulation for IceCap Flat

The RPG source code `emlFlt.rpgle` illustrates the basic setup for IceCap Flat, which is a straightforward 1:1 emulation. Although the code in this file is minimal, it demonstrates the flat emulation approach. For more advanced examples of how to manipulate the emulation, please refer to the sources for IceCap Flex and Flow.

This program utilizes the procedure `iceCapInterpretationScreen(pVts)` to access the interpretation component, further enhancing the web client experience.

```
<%
/*
-----
----- */
/* Author .....: Claus Rytter Larsen, System & Method - Marts 2023
*/
/* Compile .....: CRTICEPGM STMF('/prj/iceCap/plugins/emlFlt.rpgle') SVRID(PORTDEV)
LIBL(*SERVER) OBJLIB(ICECDEV) */
/* Compile .....: CRTICEPGM STMF('/prj/iceCap/plugins/emlFlt.rpgle') SVRID(PORTQA )
LIBL(*SERVER) OBJLIB(ICECQA) */
/* Compile .....: CRTICEPGM STMF('/prj/iceCap/plugins/emlFlt.rpgle') SVRID(PORT761)
LIBL(*SERVER) OBJLIB(ICEC761) */
/* Compile .....: CRTICEPGM STMF('/prj/iceCap/plugins/emlFlt.rpgle') SVRID(PORTQA )
LIBL(*SERVER) OBJLIB(ICECDIST) */
/*
-----
----- */
ctl-opt copyright('System & Method (C), 2021');
ctl-opt decEdit('0,') datEdit(*YMD.) main(main);
ctl-opt text('ICECAP: User Emul example');
ctl-opt bndDir('ICECAPTERM':'ICECAPEMUL':'EXTUTILITY');

/Include qrppleref,iceCapTerm
/Include qrppleref,iceCapEmul
/include qrppleref,iceCapTool
/include qasphdr,iceUtility
/include qasphdr,jsonparser
/include qasphdr,xmlparser
/include qasphdr,extUtility

/* -----
Main Line:
----- */
dcl-proc main;

    dcl-pi *n ;
        vts                likeds(vtsDS);
```

```

end-pi ;

dcl-s pVts          pointer;
dcl-s pOutput       pointer;
dcl-s extjs         varchar(32767);

setContentType('application/json; charset=utf-8');
setEncodingType('*JSON');
json_setDelimiters('/\@[ ] .{ }'"'$');

pVts = %addr(vts);

//vtSetCcsid(pVts:1141);

pOutput = emulate(pVts);

if (pOutput <> *NULL);
    emlAddMetaField(
        json_asJsonText16M(pOutput)
    );
    json_delete(pOutput);
endif;

json_delete(pVts);

return;

end-proc;

/* ----- */
/* Emulate 5250 ? */
/* ----- */
dcl-proc emulate;

    dcl-pi *n          pointer;
        pVts          pointer;
    end-pi;

    dcl-s pOutput       pointer;
    dcl-s l             int(5);
    dcl-s line          varchar(256);
    dcl-ds vts          likeds(vtsDS) based(pVts);

    for l = 1 to vts.winHeight;
        line = vtWinGetLine(pVts : l);
        if (%scan('...+..' :line) > 0);
            return *null;
        endif;
    endfor;

```

```
// Find all special component: datefield, checkbox, etc.
iceCapInterpretationScreen(pVts);

return pOutput;

end-proc;
```

Emulation for IceCap Flex

The RPG source code `emlFlx.rpgle` compared with `emlFlxCorp.rpgle` shows the typical customizations that can be made within the emulation component. Both programs use the `iceCapInterpretationScreen(pVts)` procedure to access the interpretation component, enriching the web client.

```
<%
/*
-----
----- */
/* Author .....: Claus Rytter Larsen, System & Method - Marts 2023
*/
/* Compile .....: CRTICEPGM STMF('/prj/iceCap/plugins/emlFlx.rpgle') SVRID(PORTDEV)
LIBL(*SERVER) OBJLIB(ICECDEV) */
/* Compile .....: CRTICEPGM STMF('/prj/iceCap/plugins/emlFlx.rpgle') SVRID(PORTQA )
LIBL(*SERVER) OBJLIB(ICECQA) */
/* Compile .....: CRTICEPGM STMF('/prj/iceCap/plugins/emlFlx.rpgle') SVRID(PORT761)
LIBL(*SERVER) OBJLIB(ICEC761) */
/* Compile .....: CRTICEPGM STMF('/prj/iceCap/plugins/emlFlx.rpgle') SVRID(PORTQA )
LIBL(*SERVER) OBJLIB(ICECDIST) */
/*
-----
----- */
ctl-opt copyright('System & Method (C), 2021');
ctl-opt decEdit('0,') datEdit(*YMD.) main(main);
ctl-opt text('ICECAP: User Emul example');
ctl-opt bndDir('ICECAPTERM':'ICECAPEMUL':'EXTUTILITY');

/Include qrppleref,iceCapTerm
/Include qrppleref,iceCapEmul
/include qrppleref,iceCapTool
/include qrppleref,jsonparser
/include qrppleref,xmlparser
/include qasphdr,iceUtility
/include qasphdr,extUtility

/* /Include qrppleref,iceCapTerm
/Include qrppleref,iceCapEmul
/include qrppleref,iceCapTool
/include qasphdr,iceUtility
/include qasphdr,jsonparser
/include qasphdr,xmlparser
/include qasphdr,extUtility */

/* -----
Main Line:
----- */
```

```

dcl-proc main;

    dcl-pi *n;
        vts                likeds(vtsDS);
    end-pi ;

    dcl-s pVts                pointer;
    dcl-s pOutput            pointer;
    dcl-s extjs                varchar(32767);

    setContentType('application/json; charset=utf-8');
    setEncodingType('*JSON');
    json_setDelimiters('/\@[[] .{}'"$');

    pVts = %addr(vts);

// vtSetCcsid(pVts:277);

    pOutput = emulate(pVts);

    if (pOutput <> *NULL);
        emlAddMetaField(
            json_AsJsonText16M(pOutput)
        );
        json_delete(pOutput);
    endif;

    return;

end-proc;

/* ----- */
/* Emulate 5250 ? */
/* ----- */
dcl-proc emulate;

    dcl-pi *n
        pVts                pointer;
    end-pi;

    dcl-s pOutput            pointer;
    dcl-s pParams            pointer;
    dcl-s endLine            int(5);
    dcl-s l                  int(5);
    dcl-s line                varchar(256);
    dcl-ds vts                likeds(vtsDS) based(pVts);
    dcl-ds config            likeds(emlConfigDS);
    dcl-ds keys              likeds(emlKeysDS);

    for l = 1 to vts.winHeight;

```

```
        line = vtWinGetLine(pVts : 1);
        if (%scan('...+..' :line) > 0);
            return *null;
        endif;
    endfor;

    // Find all special component: datefield, checkbox, etc.
    iceCapInterpretationScreen(pVts);

    // Find F-key line and flex from top to this line
    clear config;
    keys = emlGetKeys(
        pVts
        :config
        :''
        :vts.winHeight-2
        :vts.winHeight
        :''
        :*OFF
    );
    if (keys.strlin = 0
    or keys.strlin = 99);
        endLine = vts.winHeight;
    else;
        endLine = keys.strlin - 1;
    endif;

    pParams = json_newObject();
    json_setint(pParams : 'startLine' :1);
    json_setint(pParams : 'endLine' :endLine);
    json_setint(pParams : 'colorTheme' :reqnum('colorTheme' :0));
    pOutput = icecapCreateFlexLayout(pVts: pParams);
    json_delete(pParams);

    return pOutput;

end-proc;
```

The `emlFlxCorp.rpgle` source code is used in the demonstration environment and serves as an example of the kind of customization that can be applied to the emulation component.

```
<%
/* CMD:CRVICEPGM */
/*
-----
---- */
/* Author .....: Claus Rytter Larsen, System & Method - Marts 2023
*/
/* Compile ....: CRVICEPGM STMF('/prj/iceCap/plugins/emlFlxCorp.rpgle') SVRID(PORTDEV)
LIBL(*SERVER) OBJLIB(ICECDEV) */
/* Compile ....: CRVICEPGM STMF('/prj/iceCap/plugins/emlFlxCorp.rpgle') SVRID(PORTQA )
LIBL(*SERVER) OBJLIB(ICECQA) */
/* Compile ....: CRVICEPGM STMF('/prj/iceCap/plugins/emlFlxCorp.rpgle') SVRID(PORT761)
LIBL(*SERVER) OBJLIB(ICEC761) */
/* Compile ....: CRVICEPGM STMF('/prj/iceCap/plugins/emlFlxCorp.rpgle') SVRID(PORTQA )
LIBL(*SERVER) OBJLIB(ICECDIST) */
/*
-----
---- */
ctl-opt copyright('System & Method (C), 2021');
ctl-opt decEdit('0,') datEdit(*YMD.) main(main);
ctl-opt text('ICECAP: User Emul example');
ctl-opt bndDir('ICECAPTERM':'ICECAPEMUL':'EXTUTILITY');

/Include qrppleref,iceCapTerm
/Include qrppleref,iceCapEmul
/include qrppleref,iceCapTool
/include qrppleref,jsonparser
/include qasphdr,iceUtility
/include qasphdr,extUtility

/* -----
Main Line:
----- */
dcl-proc main;

    dcl-pi *n;
        vts                likeds(vtsDS);
    end-pi ;

    dcl-s pVts                pointer;
    dcl-s pOutput              pointer;
    dcl-s extjs                varchar(32767);

    setContentType('application/json; charset=utf-8');
    setEncodingType('*JSON');
    json_setDelimiters('/\@[[] .{}'"$');
```

```

    pVts = %addr(vts);

    // vtSetCcsid(pVts:277);

    pOutput = emulate(pVts);

    if (pOutput <> *NULL);
        emlAddMetaField(
            json_AsJsonText16M(pOutput)
        );
        json_delete(pOutput);
    endif;

    return;

end-proc;

/* ----- */
Emulate 5250 ?
/* ----- */

dcl-proc emulate;

    dcl-pi *n                pointer;
        pVts                pointer;
    end-pi;

    dcl-s pOutput            pointer;
    dcl-s pPanel             pointer;
    dcl-s pItems             pointer;
    dcl-s pItem              pointer;
    dcl-s pParams            pointer;
    dcl-s endLine            int(5);
    dcl-s l                  int(5);
    dcl-s line               varchar(256);
    dcl-s src2               varchar(1024);
    dcl-s src3               varchar(1024);
    dcl-s empName            varchar(1024);
    dcl-ds vts               likeds(vtsDS) based(pVts);
    dcl-ds config            likeds(emlConfigDS);
    dcl-ds keys              likeds(emlKeysDS);

    for l = 1 to vts.winHeight;
        line = vtWinGetLine(pVts : l);
        if (%scan('...+..' :line) > 0);
            return *null;
        endif;
    endfor;

```

```

// Find all special component: datefield, checkbox, etc.
iceCapInterpretationScreen(pVts);

// Find F-key line and flex from top to this line
clear config;
keys = emlGetKeys(
    pVts
    :config
    :''
    :vts.winHeight-2
    :vts.winHeight
    :''
    :*OFF
);
if (keys.strlin = 0
or keys.strlin = 99);
    endLine = vts.winHeight;
else;
    endLine = keys.strlin - 1;
endif;

pParams = json_newObject();
json_setint(pParams : 'startLine' :1);
json_setint(pParams : 'endLine' :endLine);
json_setint(pParams : 'colorTheme' :reqnum('colorTheme' :0));
pOutput = icecapCreateFlexLayout(pVts: pParams);
json_delete(pParams);

// Work With Employee
if (vtSaaTitleContains(pVts : 'Display employee')
or vtSaaTitleContains(pVts : 'Change employee'));

    // Add a Bing-Search for the name
    empName = vtGetValueFor(pVts : 'Name') + ' '
        + vtGetValueFor(pVts : 'Surname');

    src2 = 'http://bing.com/search?q=' + urlEncode(empName);
    src2 = (
        <iframe src='${src2}' width=100% height=100% frameborder=0 />
    );

    // Add a pdf - Job Application
    src3 = 'https://eforms.com/images/2018/03/Simple-Job-Application.pdf';
    src3 = (
        <iframe src='${src3}' width=100% height=100% frameborder=0 />
    );

    // Tap Panel
    pPanel = json_newObject();

```

```
    json_setStr(pPanel : 'xtype': 'tabpanel');
    json_setBool(pPanel : 'autoScroll': *off);
    json_setStr(pPanel : 'layout': 'fit');
    json_setInt(pPanel : 'minHeight': 800);
    json_setInt(pPanel : 'minWidth': 1100);

    pItems = json_newArray();
    json_moveObjectInto(pPanel: 'items': pItems);

    // 1st tab (employee)
    json_setStr(pOutput : 'title': 'Employee');
    json_arrayPush(pItems : pOutput);

    // 2nd tab (Internet Search)
    pItem = json_newObject();
    json_setStr(pItem : 'xtype': 'panel');
    json_setStr(pItem : 'title': 'Internet Search');
    json_setStr(pItem : 'html': %trim(src2));
    json_setInt(pItem : 'height': 1000);
    json_arrayPush(pItems : pItem);

    // 3rd tab (Job Application)
    pItem = json_newObject();
    json_setStr(pItem : 'xtype': 'panel');
    json_setStr(pItem : 'title': 'Job Application');
    json_setStr(pItem : 'html': %trim(src3));
    json_setInt(pItem : 'height': 1000);
    json_arrayPush(pItems : pItem);

    return pPanel;
endif;

return pOutput;

end-proc;
```

Emulation for IceCap Flow/Flip

The RPG source code `emlFlw.rpgle` provides a detailed technical example of how to identify and transform subfiles within the emulation process. Given the variety of ways subfiles can be designed and used, customizing this program is often necessary.

```
<%
/*
-----
----- */
/* Author .....: Claus Rytter Larsen, System & Method - Marts 2023
*/
/* Compile .....: CRTICEPGM STMF('/prj/iceCap/plugins/emlFlw.rpgle') SVRID(PORTDEV)
LIBL(*SERVER) OBJLIB(ICECDEV) */
/* Compile .....: CRTICEPGM STMF('/prj/iceCap/plugins/emlFlw.rpgle') SVRID(PORTQA)
LIBL(*SERVER) OBJLIB(ICECQA) */
/* Compile .....: CRTICEPGM STMF('/prj/iceCap/plugins/emlFlw.rpgle') SVRID(PORT761)
LIBL(*SERVER) OBJLIB(ICEC761) */
/* Compile .....: CRTICEPGM STMF('/prj/iceCap/plugins/emlFlw.rpgle') SVRID(PORTQA)
LIBL(*SERVER) OBJLIB(ICECDIST) */
/*
-----
----- */
ctl-opt copyright('System & Method (C), 2021');
ctl-opt decEdit('0,') datEdit(*YMD.) main(main);
ctl-opt text('ICECAP: User Emul example');
ctl-opt bndDir('ICECAPTERM':'ICECAPEMUL':'EXTUTILITY');

/Include qrppleref,iceCapTerm
/Include qrppleref,iceCapEmul
/include qrppleref,iceCapTool
/include qasphdr,iceUtility
/include qasphdr,jsonparser
/include qasphdr,xmlparser
/include qasphdr,extUtility

/* -----
Main Line:
----- */
dcl-proc main;

    dcl-pi *n ;
        vts                likeds(vtsDS);
    end-pi ;

    dcl-s pVts                pointer;
    dcl-s pOutput              pointer;
    dcl-s extjs                varchar(32767);
```

```

setContentType('application/json; charset=utf-8');
setEncodingType('*JSON');
json_setDelimiters('/\@[] .{}''"$');

pVts = %addr(vts);

//vtSetCcsid(pVts:1141);

pOutput = emulate(pVts);

if (pOutput <> *NULL);
    emlAddMetaField(
        json_AsJsonText16M(pOutput)
    );
    json_delete(pOutput);
endif;

return;

end-proc;

/* ----- */
Emulate 5250 ?
/* ----- */

dcl-proc emulate;

    dcl-pi *n                pointer;
        pVts                pointer;
    end-pi;

    dcl-s pOutput            pointer;
    dcl-s pParams            pointer;
    dcl-s pGrid              pointer;
    dcl-s pItems             pointer;
    dcl-s pItem              pointer;
    dcl-s gridStartLine      int(5);
    dcl-s gridEndLine        int(5);
    dcl-s keyStartLine       int(5);
    dcl-s l                  int(5);
    dcl-s line               varchar(256);
    dcl-ds vts               likeds(vtsDS) based(pVts);
    dcl-ds config            likeds(emlConfigDS);
    dcl-ds keys              likeds(emlKeysDS);
    dcl-s maxWidth           int(5);
    dcl-s minWidth            int(5);
    dcl-s colorTheme         int(5);
    dcl-s title              varchar(132);
    dcl-s src                varchar(128);
    dcl-s empName            varchar(1024);

```

```

dcl-s extCmd1          varchar(1024);
dcl-s extCmd2          varchar(1024);

colorTheme = reqnum('colorTheme' :0);

for l = 1 to vts.winHeight;
  line = vtWinGetLine(pVts : l);
  if (%scan('...+..' :line) > 0);
    return *null;
  endif;
endfor;

// Find F-key line and flex from top to this line
clear config;
keys = emlGetKeys(
  pVts
  :config
  :''
  :vts.winHeight-2
  :vts.winHeight
  :''
  :*OFF
);
if (keys.strlin = 0
or keys.strlin = 99);
  keyStartLine = vts.winHeight;
else;
  keyStartLine = keys.strlin;
endif;

title = vtWinTitle(pVts);
if (title = 'Hjælp til: IBM i-hovedmenu'           // Danish
or title = 'IBM i Main Menu - Help'                // English
or title = 'IBM i-Hauptmenü - Hilfetext'           // German
or title = 'IBM i-hoofdmenu - Help'                // Dutch
or title = 'Menu principale IBM i - Aiuto'          // Italian
or title = 'Menu principal IBM i - Aide'            // French
or title = 'Menú principal de IBM i - Ayuda'        // Spanish
or title = 'Menu Principal do IBM i - Ajuda'        // Portuguese
or title = 'Hjælp - Meny'                          // Swedish
or title = 'IBM i - Hovedmeny - hjælp'              // Norwegian
or title = 'IBM i -pæævalikko - ohje');            // Finnish
  pGrid = *NULL;
else;
  pGrid = createIceTagSubFile(pVts);

  if (pGrid = *NULL);
    pParams = setupSubfileRecognition(pVts);
  endif;
endif;

```

```

        pGrid = icecapFindAndBuildSubfile(pVts :pParams);
    endif;

    // Find all special component: datefield, checkbox, etc.
    iceCapInterpretationScreen(pVts);

    if (pGrid <> *NULL);
        minWidth = vts.winWidth * 12;
        maxWidth = minWidth;

        pOutput = json_newObject();
        json_setstr(pOutput : 'xtype' : 'panel');
        json_setstr(pOutput : 'ui' : 'icecap');
        json_setstr(pOutput : 'layout' : 'anchor');
        json_setint(pOutput : 'minWidth' : minWidth);
        json_setint(pOutput : 'maxWidth' : maxWidth);
        pItems = json_newArray();
        json_moveObjectInto(pOutput : 'items' : pItems);

        gridStartLine = json_getint(pGrid : 'startLine');
        gridEndLine = json_getint(pGrid : 'endLine');

        // Area before subfile
        pParams = json_newObject();
        json_setint(pParams : 'startLine' : 1);
        json_setint(pParams : 'endLine' : gridStartLine - 1);
        json_setint(pParams : 'colorTheme' : colorTheme);
        json_arrayPush(pItems : icecapCreateFlexLayout(pVts: pParams));
        json_delete(pParams);

        // Subfile
        json_arrayPush(pItems : pGrid);

        // Area after subfile
        pParams = json_newObject();
        json_setint(pParams : 'startLine' : gridEndLine + 1);
        json_setint(pParams : 'endLine' : keyStartLine - 1);
        json_setint(pParams : 'colorTheme' : colorTheme);
        pItem = icecapCreateFlexLayout(pVts: pParams);
        json_setstr(pItem : 'margin' : '24px 0 24px 0');
        json_arrayPush(pItems : pItem);
        json_delete(pParams);
    endif;
endif;
if (pGrid = *NULL);
    // Find all special component: datefield, checkbox, etc.
    iceCapInterpretationScreen(pVts);

    pParams = json_newObject();
    json_setint(pParams : 'startLine' : 1);

```

```

        json_setint(pParams : 'endLine' : keyStartLine - 1);
        json_setint(pParams : 'colorTheme' : colorTheme);
        pOutput = icecapCreateFlexLayout(pVts: pParams);
        json_delete(pParams);
    endif;

    return pOutput;

end-proc;

/* ----- */
Find Components
- Date fields, checkbox, etc.
/* ----- */
dcl-proc findComponents;

    dcl-pi *n;
        pVts                pointer;
    end-pi;

    dcl-s pConfig            pointer;
    dcl-s pElement           pointer;
    dcl-s pProc              pointer (*PROC) static;
    dcl-s config             varchar(256);
    dcl-s prevConfig         varchar(256) static inz('');
    dcl-s libName            char(10);
    dcl-s pgmName            char(10);
    dcl-s procName           varchar(128);
    dcl-s interpretationPgm  varchar(32) static;
    dcl-pr ActionProc        pointer extproc(pProc);
        payload             pointer;
    end-pr;

    pConfig = icecapGetIceCapConfig();

    config = reqstr('config' : '*DFT');

    if (config <> prevConfig);
        pElement = xml_locate(
            pConfig: '/config/' + %trim(config) + '/breakOut'
        );
        if (pElement = *NULL);
            pElement = xml_locate(pConfig: '/config/default/breakOut');
        endif;

        if (pElement <> *NULL);
            interpretationPgm = xml_getAttr(pElement: 'interpretationProgram:');
        endif;
    endif;
end-proc;

```

```

prevConfig = config;

if (interpretationPgm > '');
    if (words(interpretationPgm : '/') = 1);
        libName = '*LIBL';
        pgmName = interpretationPgm;
    else;
        libName = word(interpretationPgm :1 : '/');
        pgmName = word(interpretationPgm :2 : '/');
    endif;
    procName = 'ICECAPINTERPRETATION';
    pProc = loadServiceProgramProc(libName :pgmName :procName);
    if (pProc <> *NULL);
        monitor;
            ActionProc(pVts);
        on-error;
        endmon;
    endif;
endif;

return;

end-proc;

/* ----- */
Set up subfile recognition
/* ----- */
dcl-proc setupSubfileRecognition;

    dcl-pi *n                pointer;
        pVts                pointer;
    end-pi;

    dcl-s pOutput            pointer static;
    dcl-s pOptionGuide       pointer;
    dcl-s pOptionHeader      pointer;
    dcl-s pMore              pointer;
    dcl-s s                  int(10);
    dcl-s e                  int(10);
    dcl-ds vts              likeds(vtsDS) based(pVts);

    pOutput = json_newObject();

    // Set up option guide line to recognize
    pOptionGuide = json_newArray();
    /*
    json_arrayPush(
        pOptionGuide
        :setSearchValue('ANGIV VALG' :1 :vts.winWidth)

```

```

);
json_arrayPush(
    pOptionGuide
    :setSearchValue('TRYK PÅ ENTER FOR' :1 :vts.winWidth)
);
json_arrayPush(
    pOptionGuide
    :setSearchValue('TYPE OPTIONS, PRESS ENTER' :1 :vts.winWidth)
);
json_arrayPush(
    pOptionGuide
    :setSearchValue(
        'TYPE CHANGES (IF ALLOWED), PRESS ENTER'
        :1
        :vts.winWidth
    )
);
*/
json_moveObjectInto(pOutput :'optionGuide' :pOptionGuide);

// Set up option guide header to recognize
pOptionHeader = json_newArray();
json_arrayPush(pOptionHeader :setSearchValue('OPT' :1 :8));
json_arrayPush(pOptionHeader :setSearchValue('OPT.' :1 :8));
json_arrayPush(pOptionHeader :setSearchValue('OPTION' :1 :8));
json_arrayPush(pOptionHeader :setSearchValue('VLG' :1 :8));
json_arrayPush(pOptionHeader :setSearchValue('VALG' :1 :8));
json_moveObjectInto(pOutput :'optionHeader' :pOptionHeader);

// Set up subfile more and bottom
pMore = json_newArray();
s = vts.winWidth - 20;
e = vts.winWidth;
setSubfileMoreValue(pMore :'MER...' : 'SLUT' :s :e); // DK
setSubfileMoreValue(pMore :'MORE...' : 'BOTTOM' :s :e); // EN
setSubfileMoreValue(pMore :'WEITERE ...' : 'ENDE' :s :e); // DE
setSubfileMoreValue(pMore :'MEER...' : 'EINDE' :s :e); // NL
setSubfileMoreValue(pMore :'SEGUE...' : 'FINE' :s :e); // IT
setSubfileMoreValue(pMore :'A SUIVRE...' : 'FIN' :s :e); // FR
setSubfileMoreValue(pMore :'MÁS...' : 'FIM' :s :e); // ES
setSubfileMoreValue(pMore :'MAIS...' : 'FIM' :s :e); // PT
setSubfileMoreValue(pMore :'FORTS...' : 'SLUT' :s :e); // SE
setSubfileMoreValue(pMore :'MER...' : 'SLUTT' :s :e); // NO
setSubfileMoreValue(pMore :'JATKU...' : 'LOPPU' :s :e); // FI
setSubfileMoreValue(
    pMore
    : '+'
    : ' '
    : vts.winWidth-10
    : vts.winWidth

```

```

    );
    json_moveObjectInto(pOutput : 'subfileMore' : pMore);

    json_setbool(pOutput : 'showOptionColumn' : *OFF);

    return pOutput;

end-proc;

/* ----- */
  Set up search value
/* ----- */
dcl-proc setSearchValue;

    dcl-pi *n                pointer;
        text                varchar(256) value;
        start               int(10) value;
        end                 int(10) value;
    end-pi;

    dcl-s pOutput            pointer;

    pOutput = json_newObject();
    json_setstr(pOutput : 'text' : text);
    json_setint(pOutput : 'start' : start);
    json_setint(pOutput : 'end' : end);

    return pOutput;

end-proc;

/* ----- */
  Set up search value
/* ----- */
dcl-proc setSubfileMoreValue;

    dcl-pi *n;
        pSubfileMore        pointer;
        moreText             varchar(256) value;
        bottomText           varchar(256) value;
        start                int(10) value;
        end                  int(10) value;
    end-pi;

    dcl-s pOption            pointer;
    dcl-ds vts               likeds(vtsDS) based(pVts);

    pOption = json_newObject();
    json_setstr(pOption : 'moreText' : moreText);
    json_setstr(pOption : 'bottomText' : bottomText);

```

```

json_setint(pOption :'start' :start);
json_setint(pOption :'end' :end);
json_arrayPush(pSubfileMore :pOption);

```

```

return;

```

```

end-proc;

```

```

dcl-proc createIceTagSubFile;

```

```

dcl-pi *n                pointer;
      pVts                pointer value;
end-pi;

```

```

dcl-s pos                int(5);
dcl-s len                int(5);
dcl-s col                int(5);
dcl-s lin                int(5);
dcl-s off                int(5);
dcl-s attr               char(1);
dcl-s dta                varchar(256);
dcl-s component          varchar(32);
dcl-ds vtLocation        likeds(vtLocationDS);
dcl-s pGrid              pointer;

```

```

pos = VT_FIRST;
dow vtGetOutputList(pVts:pos:lin:col:off:len:attr:dta);

```

```

    component = '';

```

```

    if %scan('{#':dta) > 0
    and %scan('}':dta) > 0;
        component = 'icetag';
    endif;

```

```

    if %scan('(#':dta) > 0
    and %scan(')':dta) > 0;
        component = 'icetag';
    endif;

```

```

    if %scan('('*:dta) > 0
    and %scan(')':dta) > 0;
        component = 'icetag';
    endif;

```

```

    if %scan('(':dta) > 0
    and %scan('/':dta) > 0
    and %scan(')':dta) > 0;
        component = 'icetag';
    endif;

```

```

    if lin = 1
    and col >= 60;
        component = 'icetag';
    endif;

    if component = 'icetag';

        pGrid = setupIceTagSubFile(
            pVts
            :dta
            :lin
            :col
            :len
        );
    endif;

    if pGrid <> *NULL;
        leave;
    endif;

    pos = VT_NEXT;
enddo;

return pGrid;

end-proc;

dcl-proc setupIceTagSubFile;

    dcl-pi *n                pointer;
    pVts                    pointer value;
    dta                     varchar(256) value;
    lin                     int(5);
    col                     int(5);
    len                     int(5);
end-pi;

    dcl-s pConfig            pointer;
    dcl-s pGrid              pointer;
    dcl-s pSql               pointer;

    dcl-s sqlStmt            varchar(256);

    sqlStmt = (`
        select
        config
        from ICETAG
        WHERE ID = '${dta}'
        and lower(type) = 'subfile'

```

```
        and active = 1
    `);

    pSql = json_sqlResultRow(sqlStmt);

    if (pSql = *NULL);
        return *NULL;
    else;
        pConfig = json_locate(pSql: 'config');
        pGrid = icecapArea2Grid(
            pVts
            :pConfig
        );

        vtwinPatchRaw(pVts: lin :col : '' : len);
    endif;

    return pGrid;

end-proc;
```

Additionally, the `emlFlwCorp.rpgle` source code demonstrates how to embed extra tabs and charts into the emulation. The latter part of this code is particularly valuable, as it provides practical examples of methods and syntax. By comparing it with the standard source, the specific customizations become evident.

```

/*
----- */
/* Author .....: Claus Rytter Larsen, System & Method - Marts 2023
*/
/* Compile DEV : CRTICEPGM STMF('/prj/iceCap/plugins/emlFlwCorp.rpgle') SVRID(PORTDEV)
LIBL(*SERVER) */
/* Compile QA : CRTICEPGM STMF('/prj/iceCap/plugins/emlFlwCorp.rpgle') SVRID(PORTQA )
LIBL(*SERVER) */
/* Compile PROD: CRTICEPGM STMF('/prj/iceCap/plugins/emlFlwCorp.rpgle') SVRID(PORT761)
LIBL(*SERVER) */
/* Compile DIST: CRTICEPGM STMF('/prj/iceCap/plugins/emlFlwCorp.rpgle') SVRID(PORTQA )
LIBL(*SERVER) */
/*
----- */
ctl-opt copyright('System & Method (C), 2021');
ctl-opt decEdit('0,') datEdit(*YMD.) main(main);
ctl-opt text('ICECAP: User Emul example');
ctl-opt bndDir('ICECAPTERM':'ICECAPEMUL':'EXTUTILITY');

/Include qrpgleref,iceCapTerm
/Include qrpgleref,iceCapEmul
/include qrpgleref,iceCapTool
/include qasphdr,iceUtility
/include qasphdr,jsonparser
/include qasphdr,xmlparser
/include qasphdr,extUtility

/* -----
Main Line:
----- */
dcl-proc main;

    dcl-pi *n ;
        vts                likeds(vtsDS);
    end-pi ;

    dcl-s pVts                pointer;
    dcl-s pOutput              pointer;
    dcl-s extjs                varchar(32767);

    setContentType('application/json; charset=utf-8');
```

```

setEncodingType('*JSON');
json_setDelimiters('/\@[] .{}' '$');

pVts = %addr(vts);

pOutput = emulate(pVts);

if (pOutput <> *NULL);
    emlAddMetaField(
        json_asjsonText16m(pOutput)
    );
    json_delete(pOutput);
endif;

return;

end-proc;

/* ----- */
Emulate 5250 ?
/* ----- */

dcl-proc emulate;

    dcl-pi *n                pointer;
        pVts                pointer;
    end-pi;

    dcl-s pOutput            pointer;
    dcl-s pParams            pointer;
    dcl-s pGrid              pointer;
    dcl-s pItem              pointer;
    dcl-s pItems             pointer;
    dcl-s pPanel             pointer;

    dcl-s gridStartLine      int(5);
    dcl-s gridEndLine        int(5);
    dcl-s keyStartLine       int(5);
    dcl-s l                  int(5);
    dcl-s line                varchar(256);
    dcl-s maxWidth           int(5);
    dcl-s minWidth           int(5);
    dcl-s colorTheme         int(5);
    dcl-s title              varchar(132);

    dcl-ds vts               likeds(vtsDS) based(pVts);
    dcl-ds config            likeds(emlConfigDS);
    dcl-ds keys              likeds(emlKeysDS);

    colorTheme = reqnum('colorTheme' :0);

```

```

for l = 1 to vts.winHeight;
    line = vtWinGetLine(pVts : l);
    if (%scan('...+..' :line) > 0);
        return *null;
    endif;
endfor;

// ----- //
// Intepretation and manipulation //
// ----- //
// Beef up design with special components: datefield, checkbox, combo boxes
// Search utilities, pretype
// ----- //
iceCapInterpretationScreen(pVts);

// Find F-key Line and flex from top to this line
clear config;
keys = emlGetKeys(
    pVts
    :config
    :''
    :vts.winHeight-3
    :vts.winHeight
    :''
    :*OFF
);
if (keys.strlin = 0
or keys.strlin = 99);
    keyStartLine = vts.winHeight;
else;
    keyStartLine = keys.strlin;
endif;

// Grab Title - Often/Normally White Text In Line 1
title = vtWinTitle(pVts);
if (title = 'Hjælp til: IBM i-hovedmenu' // Danish
or title = 'IBM i Main Menu - Help' // English
or title = 'IBM i-Hauptmenü - Hilfetext' // German
or title = 'IBM i-hoofdmenu - Help' // Dutch
or title = 'Menu principale IBM i - Aiuto' // Italian
or title = 'Menu principal IBM i - Aide' // French
or title = 'Menú principal de IBM i - Ayuda' // Spanish
or title = 'Menu Principal do IBM i - Ajuda' // Portuguese
or title = 'Hjælp - Meny' // Swedish
or title = 'IBM i - Hovedmeny - hjelp' // Norwegian
or title = 'IBM i -pæævalikko - ohje'); // Finnish
    pGrid = *NULL;
else;
    pParams = setupSubfileRecognition(pVts);
    pGrid = icecapFindAndBuildSubfile(pVts :pParams);

```

```

if (pGrid <> *NULL);
    minWidth = vts.winWidth * 10 * 1.125;
    maxWidth = minWidth * 1.125;

    pOutput = json_newObject();
    json_setstr(pOutput : 'xtype' : 'panel');
    json_setstr(pOutput : 'ui' : 'icecap');
    json_setstr(pOutput : 'layout' : 'anchor');
    json_setint(pOutput : 'minWidth' : minWidth);
    json_setint(pOutput : 'maxWidth' : maxWidth);
    pItems = json_newArray();
    json_moveObjectInto(pOutput : 'items' : pItems);

    gridStartLine = json_getint(pGrid : 'startLine');
    gridEndLine = json_getint(pGrid : 'endLine');

    // Area before subfile
    pParams = json_newObject();
    json_setint(pParams : 'startLine' : 1);
    json_setint(pParams : 'endLine' : gridStartLine - 1);
    json_setint(pParams : 'colorTheme' : colorTheme);
    json_arrayPush(pItems : icecapCreateFlexLayout(pVts: pParams));
    json_delete(pParams);

    // Subfile
    json_arrayPush(pItems : pGrid);

    // Area after subfile
    pParams = json_newObject();
    json_setint(pParams : 'startLine' : gridEndLine + 1);
    json_setint(pParams : 'endLine' : keyStartLine - 1);
    json_setint(pParams : 'colorTheme' : colorTheme);
    pItem = icecapCreateFlexLayout(pVts: pParams);
    json_setstr(pItem : 'margin' : '24px 0 24px 0');
    json_arrayPush(pItems : pItem);
    json_delete(pParams);
endif;
endif;

// Not grid (Subfile)
if (pGrid = *NULL);
    pParams = json_newObject();
    json_setint(pParams : 'startLine' : 1);
    json_setint(pParams : 'endLine' : keyStartLine - 1);
    json_setint(pParams : 'colorTheme' : colorTheme);

    pOutput = icecapCreateFlexLayout(pVts: pParams);
    json_delete(pParams);

    // Work With Employee

```

```

    // Add Additional 'tabs' for Internet Search and Job application to screen.
    if (vtSaaTitleContains(pVts : 'Display employee')
    or vtSaaTitleContains(pVts : 'Change employee'));

        // Create additional "Tab Panels"
        pPanel = create_panels(pVts:pOutput);

        return pPanel;

    endif;
endif;

return pOutput;

end-proc;

/* ----- */
Find Components
- Date fields, checkbox, etc.
/* ----- */
dcl-proc findComponents;

    dcl-pi *n;
        pVts                pointer;
    end-pi;

    dcl-s pConfig            pointer;
    dcl-s pElement           pointer;
    dcl-s pProc              pointer (*PROC) static;
    dcl-s config             varchar(256);
    dcl-s prevConfig         varchar(256) static inz('');
    dcl-s libName            char(10);
    dcl-s pgmName            char(10);
    dcl-s procName           varchar(128);
    dcl-s interpretationPgm  varchar(32) static;
    dcl-pr ActionProc        pointer extproc(pProc);
        payload             pointer;
    end-pr;

    pConfig = icecapGetIceCapConfig();

    config = reqstr('config' : '*DFT');

    if (config <> prevConfig);
        pElement = xml_locate(
            pConfig: '/config/' + %trim(config) + '/breakOut'
        );
        if (pElement = *NULL);
            pElement = xml_locate(pConfig: '/config/default/breakOut');
        endif;
    endif;
end-proc;

```

```

endif;

if (pElement <> *NULL);
    interpretationPgm = xml_getAttr(pElement:'interpretationProgram:');
endif;
endif;

prevConfig = config;

if (interpretationPgm > '');
    if (words(interpretationPgm :'/') = 1);
        libName = '*LIBL';
        pgmName = interpretationPgm;
    else;
        libName = word(interpretationPgm :1 :'/');
        pgmName = word(interpretationPgm :2 :'/');
    endif;
    procName = 'ICECAPINTERPRETATION';
    pProc = loadServiceProgramProc(libName :pgmName :procName);
    if (pProc <> *NULL);
        monitor;
        ActionProc(pVts);
        on-error;
        endmon;
    endif;
endif;

return;

end-proc;

/* ----- */
Set up subfile recognition
/* ----- */
dcl-proc setupSubfileRecognition;

    dcl-pi *n                pointer;
        pVts                pointer;
    end-pi;

    dcl-s  pOutput            pointer static;
    dcl-s  pOptionGuide       pointer;
    dcl-s  pOptionHeader      pointer;
    dcl-s  pSubfileMore       pointer;
    dcl-ds vts                liked(vtsDS) based(pVts);

    pOutput = json_newObject();

    // Set up option guide line to recognize
    pOptionGuide = json_newArray();

```

```
/*
json_arrayPush(
    pOptionGuide
    :setSearchValue('ANGIV VALG' :1 :vts.winWidth)
);
json_arrayPush(
    pOptionGuide
    :setSearchValue('TRYK PÅ ENTER FOR' :1 :vts.winWidth)
);
json_arrayPush(
    pOptionGuide
    :setSearchValue('TYPE OPTIONS, PRESS ENTER' :1 :vts.winWidth)
);
json_arrayPush(
    pOptionGuide
    :setSearchValue(
        'TYPE CHANGES (IF ALLOWED), PRESS ENTER'
        :1
        :vts.winWidth
    )
);
*/
json_moveObjectInto(pOutput : 'optionGuide' : pOptionGuide);

// Set up option guide header to recognize
pOptionHeader = json_newArray();
json_arrayPush(pOptionHeader : setSearchValue('OPT' :1 :8));
json_arrayPush(pOptionHeader : setSearchValue('OPT.' :1 :8));
json_arrayPush(pOptionHeader : setSearchValue('OPTION' :1 :8));
json_arrayPush(pOptionHeader : setSearchValue('VLG' :1 :8));
json_arrayPush(pOptionHeader : setSearchValue('VALG' :1 :8));
json_moveObjectInto(pOutput : 'optionHeader' : pOptionHeader);

// Set up subfile more and bottom
pSubfileMore = json_newArray();
json_arrayPush(
    pSubfileMore
    :setSubfileMoreValue(
        'MORE...'
        : 'SLUT'
        :vts.winWidth-20
        :vts.winWidth
    )
);
json_arrayPush(
    pSubfileMore
    :setSubfileMoreValue(
        'MORE...'
        : 'BOTTOM'
        :vts.winWidth-20
    )
);
```

```

        :vts.winWidth
    )
);
json_arrayPush(
    pSubfileMore
    :setSubfileMoreValue(
        'WEITERE'
        : 'ENDE'
        :vts.winWidth-20
        :vts.winWidth
    )
);
json_arrayPush(
    pSubfileMore
    :setSubfileMoreValue('+ ' : ' ' :vts.winWidth-10 :vts.winWidth)
);
json_moveObjectInto(pOutput 'subfileMore' :pSubfileMore);

return pOutput;

end-proc;

/* ----- */
Set up search value
/* ----- */
dcl-proc setSearchValue;

    dcl-pi *n            pointer;
        text            varchar(256) value;
        start           int(10) value;
        end              int(10) value;
    end-pi;

    dcl-s pOutput        pointer;

    pOutput = json_newObject();
    json_setstr(pOutput 'text' :text);
    json_setint(pOutput 'start' :start);
    json_setint(pOutput 'end' :end);

    return pOutput;

end-proc;

/* ----- */
Set up search value
/* ----- */
dcl-proc setSubfileMoreValue;

    dcl-pi *n            pointer;

```

```

        moreText          varchar(256) value;
        bottomText        varchar(256) value;
        start              int(10) value;
        end                int(10) value;
    end-pi;

    dcl-s  pOutput          pointer;

    pOutput = json_newObject();
    json_setStr(pOutput : 'moreText' : moreText);
    json_setStr(pOutput : 'bottomText' : bottomText);
    json_setInt(pOutput : 'start' : start);
    json_setInt(pOutput : 'end' : end);

    return pOutput;

end-proc;

/* ----- */
Get the field-information
- and find out if it's an output/protected field
/* ----- */
dcl-proc create_panels;

    dcl-pi *n          pointer;
        pVts          pointer;
        p5250         pointer;
    end-pi;

    dcl-s  pTabPanel    pointer;
    dcl-s  pPanels      pointer;
    dcl-s  pPanelItem   pointer;

    // Create Tab Panel
    pTabPanel = json_newObject();
    json_setStr(pTabPanel : 'xtype' : 'tabpanel');
    json_setBool(pTabPanel : 'autoScroll' : *off);
    json_setStr(pTabPanel : 'layout' : 'fit');
    json_setInt(pTabPanel : 'minHeight' : 800);
    json_setInt(pTabPanel : 'minWidth' : 1100);
    pPanels = json_newArray();
    json_moveObjectInto(pTabPanel : 'items' : pPanels);

    // Tab #1 - Main 5250 + SidePanel/chart
    json_arrayPush(pPanels : create_sidepanel(pVts:p5250));

    // Tab #2 - Internet Search
    pPanelItem = create_panel_internetsearch(pVts);
    json_arrayPush(pPanels : pPanelItem);

```

```

    // Tab #3 - A PDF File
    pPanelItem = create_panel_pdf(pVts);
    json_arrayPush(pPanels :pPanelItem);

    return pTabPanel;

end-proc;

/* ----- */
    Get the field-information
    - and find out if it's an output/protected field
/* ----- */
dcl-proc create_panel_internetsearch;

    dcl-pi *n                pointer;
        pVts                pointer;
        scrollable           ind value options(*nopass);
    end-pi;

    dcl-s    w_Scrollable    ind;
    dcl-s    pPanelItem      pointer;
    dcl-s    searchE         varchar(1024);
    dcl-s    empName         varchar(1024);

    w_Scrollable = *off;
    if %parms() >= 2;
        w_Scrollable = scrollable;
    endif;

    // Add a Bing-Search for the name
    empName = vtGetValueFor(pVts : 'Name') + ' '
        + vtGetValueFor(pVts : 'Surname');
    searchE = 'http://bing.com/search?q=' + urlEncode(empName);
    searchE = (
        <iframe src='${searchE}' width=100% height=100% frameborder=0 />
    );

    // Internet Search
    pPanelItem = json_newObject();
    json_setStr(pPanelItem : 'xtype': 'panel');
    json_setInt(pPanelItem : 'height': 1000);
    json_setbool(pPanelItem : 'autoScroll': w_Scrollable);
    json_setbool(pPanelItem : 'fit': *on);
    json_setStr(pPanelItem : 'title': 'Internet Search');
    json_setStr(pPanelItem : 'html': %trim(searchE));

    return pPanelItem;

end-proc;

```

```

/* ----- */
  Get the field-information
  - and find out if it's an output/protected field
/* ----- */
dcl-proc create_panel_pdf;

  dcl-pi *n                pointer;
        pVts              pointer;
        scrollable         ind value options(*nopass);
  end-pi;

  dcl-s  pPanelItem        pointer;
  dcl-s  pdfFil            varchar(1024);
  dcl-s  w_Scrollable      ind;

  w_Scrollable = *off;
  if %parms() >= 2;
    w_Scrollable = scrollable;
  endif;

  // 3rd tab (Job Application)
  // Add a pdf - Job Application
  pdfFil = 'https://eforms.com/images/2018/03/Simple-Job-Application.pdf';
  pdfFil = (`
    <iframe src='${pdfFil}' width=100% height=100% frameborder=0 />
  `);

  pPanelItem = json_newObject();
  json_setStr(pPanelItem : 'xtype' : 'panel');
  json_setInt(pPanelItem : 'height' : 1000);
  json_setbool(pPanelItem : 'autoScroll' : w_Scrollable);
  json_setbool(pPanelItem : 'fit' : *on);
  json_setStr(pPanelItem : 'title' : 'Job Application');
  json_setStr(pPanelItem : 'html' : %trim(pdfFil));

  return pPanelItem;

end-proc;

/* ----- */
  Get the field-information
  - and find out if it's an output/protected field
/* ----- */
dcl-proc create_sidepanel;

  dcl-pi *n                pointer;
        pVts              pointer;
        p5250             pointer;
  end-pi;

```

```

dcl-s    pChart          pointer;
dcl-s    pItem           pointer;
dcl-s    pItemS          pointer;
dcl-s    pHbox           pointer;
dcl-s    pLayout         pointer;
dcl-s    pPanel          pointer;
dcl-s    pPanelItems     pointer;

// Main Panel
pHbox = json_newObject();
json_setStr(pHbox : 'title' : 'Employee');
json_setStr(pHbox : 'xtype' : 'panel');
pLayout = json_newObject();
json_setStr(pLayout : 'type' : 'hbox');
json_setStr(pLayout : 'align' : 'stretch');
json_moveObjectInto(pHbox : 'layout' : pLayout);

pPanel = json_newObject();
json_setStr(pPanel : 'xtype' : 'panel');
json_setInt(pPanel : 'width' : 900);

// hbox Left -> 5250
pPanelItems = json_newArray();
json_moveObjectInto(pPanel : 'items' : pPanelItems);
json_arrayPush(pPanelItems : p5250);

// hbox right -> chart
pItems = json_newArray();
json_moveObjectInto(pHbox : 'items' : pItems);
json_arrayPush(pItems : pPanel);

// Create -> Push Chart
pChart = create_chart(pVts);
if pChart <> *null;
    json_arrayPush(pItems : pChart);
endif;

return pHbox;

end-proc;

/* ----- */
    Get the field-information
    - and find out if it's an output/protected field
/* ----- */
dcl-proc create_chart;
```

```

dcl-pi *n          pointer;
      pVts         pointer;
end-pi;

dcl-s  pValue       pointer;
dcl-s  pChart       pointer;
dcl-s  pLegend      pointer;
dcl-s  pWrapper     pointer;
dcl-s  pWrapperItems pointer;
dcl-s  pWrapperLayout pointer;
dcl-s  pRight       pointer;
dcl-s  pStore       pointer;
dcl-s  pSeries      pointer;
dcl-s  pTemp        pointer;
dcl-s  pData        pointer;
dcl-s  pCaptions   pointer;
dcl-s  pTitle       pointer;
dcl-s  pStyle       pointer;
dcl-s  salary       int(10);
dcl-s  bonus        int(10);
dcl-s  commission   int(10);

// Add floating chart component image

// Title = vtWinTitle(pVts);
if vtSaaTitleContains(pVts : 'Display employee')
or vtSaaTitleContains(pVts : 'Change employee');
    pData = json_newArray();
    pValue = json_newObject();
    json_setstr(pValue: 'text': 'Commission');
    json_setint(pValue: 'value': %int(vtGetValueFor(pVts: 'Commission')));
    json_arrayPush(pData: pValue);
    pValue = json_newObject();
    json_setstr(pValue: 'text': 'Bonus');
    json_setint(pValue: 'value': %int(vtGetValueFor(pVts: 'Bonus')));
    json_arrayPush(pData: pValue);
    pValue = json_newObject();
    json_setint(pValue: 'value': %int(vtGetValueFor(pVts: 'Salary')));
    json_setstr(pValue: 'text': 'Salary');
    json_arrayPush(pData: pValue);

    pChart = json_newObject();
    json_setStr(pChart: 'xtype': 'polar');
    json_setBool(pChart: 'border': *on);
    json_setStr(pChart: 'margin': '60px 60px 60px 0px');
    json_setint(pChart: 'flex': 1);
    json_setint(pChart: 'innerPadding': 25);

    pCaptions = json_newObject();

```

```
pStyle = json_newObject();
json_setStr(pStyle:'fontFamily':'Open sans');
json_setStr(pStyle:'fontSize':'13px');
json_setStr(pStyle:'fontWeight':'bold');

pTitle = json_newObject();
json_setStr(pTitle:'text':'Salary Split');
json_setStr(pTitle:'align':'center');
json_setInt(pTitle:'padding':15);

json_moveObjectInto(pTitle:'style':pStyle);
json_moveObjectInto(pCaptions:'title':pTitle);

// --> panel
pStore = json_newObject();

json_moveObjectInto(pStore:'data':pData);
json_moveObjectInto(pChart:'store':pStore);

// --> legend
pLegend = json_newObject();
json_setStr(pLegend:'docked':'bottom');
json_setInt(pLegend:'padding':15);
json_moveObjectInto(pChart:'legend':pLegend);

// --> captions
json_moveObjectInto(pChart:'captions':pCaptions);

// --> series
pSeries = json_newObject();
json_setStr(pSeries:'type':'pie');
json_setStr(pSeries:'angleField':'value');
json_setBool(pSeries:'highlight':*on);

pTemp = json_newObject();
json_setStr(pTemp:'field':'text');
json_setStr(pTemp:'display':'insideEnd');
json_moveObjectInto(pSeries:'label':pTemp);

pTemp = json_newObject();
json_setDec(pTemp:'opacity':0.6);
json_moveObjectInto(pSeries:'style':pTemp);

// --> panel
json_moveObjectInto(pChart:'series':pSeries);

pWrapper = json_newObject();
json_setStr(pWrapper:'xtype':'panel');
json_setInt(pWrapper:'flex':1);
```

```
pWrapperLayout = json_newObject();
json_setStr(pWrapperLayout : 'type': 'hbox');
json_setStr(pWrapperLayout : 'align': 'stretch');
json_moveObjectInto(pWrapper: 'layout': pWrapperLayout);

pWrapperItems = json_newArray();
json_moveObjectInto(pWrapper: 'items': pWrapperItems);
json_arrayPush(pWrapperItems: pChart);

pRight = json_newObject();
json_setStr(pRight : 'xtype': 'panel');
json_setInt(pRight : 'flex': 1);

json_arrayPush(pWrapperItems: pRight);

return pChart;
endif;

end-proc;
```

Interpretation

The RPG source code `intibm.rpgle` showcases how to enhance the frontend of standard systems with a unified layout similar to IBM i OS. This program focuses on enriching keyword values when prompting IBM i commands.

```
<%@ pgmtype="srvpgm" pgmopt="EXPORT(*SRCFILE) srcfile(*LIBL/QSRVSRC) SRCMBR(INTEIBMI) %><%
/* CMD:CRVICEPGM */
/* -----
*/
/* Author .....: Claus Rytter Larsen, System & Method - Marts 2023
*/
/* Compile .....: CRVICEPGM STMF('/prj/iceCap/plugins/intIbm.rpgle') SVRID(PORTDEV)
LIBL(*SERVER) OBJLIB(ICECDEV) */
/* Compile .....: CRVICEPGM STMF('/prj/iceCap/plugins/intIbm.rpgle') SVRID(PORTQA )
LIBL(*SERVER) OBJLIB(ICECQA) */
/* Compile .....: CRVICEPGM STMF('/prj/iceCap/plugins/intIbm.rpgle') SVRID(PORT761)
LIBL(*SERVER) OBJLIB(ICEC761) */
/* Compile .....: CRVICEPGM STMF('/prj/iceCap/plugins/intIbm.rpgle') SVRID(PORTQA )
LIBL(*SERVER) OBJLIB(ICECDIST) */
/* -----
*/
ctl-opt copyright('System & Method (C), 2023');
ctl-opt decEdit('0,') datEdit(*YMD.) nomain;
ctl-opt text('ICECAP: User Emul example');
ctl-opt bndDir('ICECAPTERM':'ICECAPEMUL':'EXTUTILITY');

/Include qrppleref,iceCapTerm
/Include qrppleref,iceCapEmul
/include qrppleref,iceCapTool
/include qasphdr,iceUtility
/include qasphdr,jsonparser
/include qasphdr,extUtility

/* ----- *\\
Find Components
- Date fields, checkbox, etc.
\\* ----- */
dcl-proc iceCapInterpretation export;

dcl-pi *n;
pVts pointer;
end-pi;

dcl-ds loc likeds(vtLocationDS);
dcl-s pComponent pointer;
dcl-s pOptions pointer;
dcl-s pSql pointer;
```

```

dcl-s i                int(10);
dcl-s sqlStmt          varchar(8192);
dcl-ds vts             likeds(vtsDS) based(pVts);
dcl-ds vtField         likeds(vtFieldDS);
dcl-ds emlField        likeds(emlFieldDS);
dcl-s pos              int(5);
dcl-s len              int(5);
dcl-s col              int(5);
dcl-s lin              int(5);
dcl-s off              int(5);
dcl-s atr              char(1);
dcl-s dta              varchar(256);

// Set combobox for "job description"
sqlStmt = '';
loc = vtLocate(
    pVts
    :VT_INPUT
    :vtLinCol(1:1)
    :VT_EAST
    :'job description'
);
dow (loc > *LOVAL );
    if (sqlStmt = '');
        sqlStmt = (
            with rows as (
                select
                    objname as "value",
                    objname concat ' - ' concat objtext as "text"
                from table (QSYS2.OBJECT_STATISTICS(
                    case when ':dependency' = ''
                        then '*ALL' else ':dependency' end,
                    '*JOB'
                )
            )
            select *
            from rows
            where "value" like upper(':query%')
            order by "value"
            limit :limit offset :start
        );
        pSql = json_parseString(extSql('{}' :sqlStmt));
    endif;
    pComponent = json_newObject();
    json_setstr(pComponent : 'xtype' : 'DDSQL');
    json_setint(pComponent : 'flex' : 10);
    json_setbool(pComponent : 'hideTrigger' : *ON);
    json_setStr(pComponent : 'valueField' : 'value');
    json_setStr(pComponent : 'displayField' : 'text');

```

```

pOptions = json_newObject();
json_setint(pOptions : 'minWidth' : 300);
json_moveObjectInto(pComponent : 'listConfig' : pOptions);
pOptions = json_newArray();
json_arrayPush(pOptions : 'value');
json_moveObjectInto(pComponent : 'displayList' : pOptions);
json_copyValue(pComponent : 'sqlVoucher' : pSql : 'sqlVoucher');
vtSetMetaData(
    pVts
    :loc
    :json_asJsonText16M(pComponent)
);
json_delete(pComponent);
loc = vtLocate(pVts: VT_INPUT: loc : VT_EAST: 'job description');
enddo;
if (sqlStmt > '');
    json_delete(pSql);
endif;

// Set combobox for "job queue"
sqlStmt = '';
loc = vtLocate(
    pVts
    :VT_INPUT
    :vtLinCol(1:1)
    :VT_EAST
    : 'job queue'
);
dow (loc > *LOVAL );
    if (sqlStmt = '');
        sqlStmt = (
            with rows as (
                select
                    objname as "value",
                    objname concat ' - ' concat objtext as "text"
                from table (QSYS2.OBJECT_STATISTICS(
                    case when ':dependency' = ''
                        then '*ALL' else upper(':dependency') end,
                    '*JOBQ'
                )
            )
            select *
            from rows
            where "value" like upper(':query%')
            order by "value"
            limit :limit offset :start
        );
        pSql = json_parseString(extSql('{}' : sqlStmt));
    endif;

```

```

// Get field and field no
emlField = emlGetField(pVts :loc);
pComponent = json_newObject();
json_setstr(pComponent : 'xtype' : 'DDSQL');
json_setint(pComponent : 'flex' : 10);
json_setbool(pComponent : 'hideTrigger' : *ON);
json_setStr(pComponent : 'valueField' : 'value');
json_setStr(pComponent : 'displayField' : 'text');
json_setstr(pComponent : 'dependency' : 'i' + %char(emlField.fieldNo));
pOptions = json_newObject();
json_setint(pOptions : 'minWidth' : 300);
json_moveObjectInto(pComponent : 'listConfig' : pOptions);
pOptions = json_newArray();
json_arrayPush(pOptions : 'value');
json_moveObjectInto(pComponent : 'displayList' : pOptions);
json_copyValue(pComponent : 'sqlVoucher' : pSql : 'sqlVoucher');
vtSetMetaData(
    pVts
    :loc
    :json_asJsonText16M(pComponent)
);
json_delete(pComponent);
loc = vtLocate(pVts: VT_INPUT: loc : VT_EAST: 'job queue');
enddo;
if (sqlStmt > '');
    json_delete(pSql);
endif;

// Set combobox for "Library"
sqlStmt = '';
loc = vtLocate(
    pVts
    :VT_INPUT
    :vtLinCol(1:1)
    :VT_EAST
    : 'library'
);
dow (loc > *LOVAL );
    vtGetField(pVts :loc :vtField);
    if (%scan('library' :lowercase(vtField.value)) = 0);
        if (sqlStmt = '');
            sqlStmt = (
                with rows as (
                    select
                        objname as "value",
                        case when (objtext is null) then
                            objname
                        else
                            objname concat ' - ' concat objtext
                        end as "text"

```

```

        from table (QSYS2.OBJECT_STATISTICS('*ALL', '*LIB') )
    )
    select *
    from rows
    where "value" like upper(':query%')
    order by "value"
    limit :limit offset :start
`);
endif;
pSql = json_parseString(extSql('{}' :sqlStmt));
pComponent = json_newObject();
json_setstr(pComponent : 'xtype' : 'DDSQL');
json_setint(pComponent : 'flex' : 10);
json_setbool(pComponent : 'hideTrigger' : *ON);
json_setStr(pComponent : 'valueField' : 'value');
json_setStr(pComponent : 'displayField' : 'text');
pOptions = json_newObject();
json_setint(pOptions : 'minWidth' : 300);
json_moveObjectInto(pComponent : 'listConfig' : pOptions);
pOptions = json_newArray();
json_arrayPush(pOptions : 'value');
json_moveObjectInto(pComponent : 'displayList' : pOptions);
json_copyValue(pComponent : 'sqlVoucher' : pSql : 'sqlVoucher');
vtSetMetaData(
    pVts
    :loc
    :json_asJsonText16M(pComponent)
);
json_delete(pComponent);
endif;
loc = vtLocate(pVts: VT_INPUT: loc : VT_EAST: 'library');
enddo;
if (sqlStmt > '');
    json_delete(pSql);
endif;

// Set linkfield for https://
for i = 1 to vts.inputFields;
    vtWinGetFieldNo(pVts:i:vtField);
    if (%scan('www' :vtField.value) > 0
    or %scan('http://' :vtField.value) > 0
    or %scan('https://' :vtField.value) > 0
    or %scan('ftp://' :vtField.value) > 0);
        loc.lin = vtField.lin;
        loc.col = vtField.col;
        pComponent = json_newObject();
        json_setstr(pComponent : 'xtype' : 'LINKFIELD');
        vtSetMetaData(
            pVts
            :loc

```

```
                :json_asJsonText16M(pComponent)
            );
            json_delete(pComponent);
        endif;
    endfor;
    pos = VT_FIRST;
    dow vtGetOutputList(pVts:pos:lin:col:off:len:atr:dta);
        if (%scan('www' :dta) > 0
            or %scan('http://' :dta) > 0
            or %scan('https://' :dta) > 0
            or %scan('ftp://' :dta) > 0);
            pComponent = json_newObject();
            json_setstr(pComponent : 'xtype' : 'LINKFIELD');
            json_setBool(pComponent : 'readOnly' : *ON);
            emlAddMetaOutputField(
                lin
                :col
                :json_asJsonText16M(pComponent)
            );
            json_delete(pComponent);
        endif;
        pos = VT_NEXT;
    enddo;

    return;

end-proc;
```

The `inttag.rpgle` source code bridges Tag definitions with the interpretation component. It reveals the majority of web components ('xtype') used and documents the syntax and functionality.

```
<%@ pgmtype="srvpgm" pgmopt="EXPORT(*SRCFILE) srcfile(*LIBL/QSRVSRV) SRCMBR(INTTAG) %><%
/* CMD:CRVICEPGM */
ctl-opt copyright('System & Method (C), 2023');
ctl-opt decEdit('0,') datEdit(*YMD.) nomain;
ctl-opt text('ICECAP: Corp. Data Interpretation');
ctl-opt bndDir('ICECAPTERM':'ICECAPEMUL':'EXTUTILITY');

/*
----- */
/* Author .....: Jørn Holm / Peter Nøbbe, System & Method - 2024 - January
*/
/* Function ....: Interpretation of 5250 using IceCap for Corp. Data programs
*/
/*
*/
/* Compile ....: CRVICEPGM STMF('/prj/iceCap/plugins/intTag.rpgle') SVRID(PORTDEV)
LIBL(*SERVER) OBJLIB(ICECDEV) */
/* Compile ....: CRVICEPGM STMF('/prj/iceCap/plugins/intTag.rpgle') SVRID(PORTQA )
LIBL(*SERVER) OBJLIB(ICECQA) */
/* Compile ....: CRVICEPGM STMF('/prj/iceCap/plugins/intTag.rpgle') SVRID(PORT761)
LIBL(*SERVER) OBJLIB(ICEC761) */
/* Compile ....: CRVICEPGM STMF('/prj/iceCap/plugins/intTag.rpgle') SVRID(PORTQA )
LIBL(*SERVER) OBJLIB(ICECDIST) */
/*
----- */

/include qrppleref,iceCapTerm
/include qrppleref,iceCapEmul
/include qrppleref,iceCapTool
/include qasphdr,iceUtility
/include qasphdr,jsonparser
/include qasphdr,extUtility

dcl-c FIELD_IS_HIDDEN CONST(x'27');

/* ----- */
/* Find Components
   - Date fields, checkbox, etc.
   \* ----- */
dcl-proc iceCapInterpretation export;

    dcl-pi *n;
        pVts                pointer;
    end-pi;
```

```

        findComponents(pVts);

        return;

end-proc;

/* ----- */
Find Components
- Date fields
/* ----- */
dcl-proc findComponents;

    dcl-pi *n;
        pVts                pointer value;
    end-pi;

    dcl-s pos                int(5);
    dcl-s len                int(5);
    dcl-s col                int(5);
    dcl-s start_col         int(5);
    dcl-s end_col           int(5);
    dcl-s pad_len           int(5);
    dcl-s lin               int(5);
    dcl-s off               int(5);
    dcl-s offset            int(5);
    dcl-s attr              char(1);
    dcl-s dta               varchar(256);
    dcl-s dtaNext           varchar(256);
    dcl-s wDta              varchar(256);
    dcl-s chars             varchar(5);
    dcl-s component         varchar(32);
    dcl-s removeText        ind;
    dcl-s ibmi              ind;
    dcl-ds vtLocation       likeds(vtLocationDS);
    dcl-ds vtField          likeds(vtFieldDS);
    dcl-ds last_field       likeds(vtFieldDS);
    dcl-ds last_loc         likeds(vtLocationDS);

    // Check all output fields
    pos = VT_FIRST;
    dow vtGetOutputList(pVts:pos:lin:col:off:len:attr:dta);

        // dtaNext = getNextDta(pVts: pos: dta);

        if emlCheckSkipOutput(lin:col:len);
            pos = VT_NEXT;
            iter;
        endif;

```

```
component = '';
removeText = *OFF;
ibmi = *OFF;

dta = %trim(dta);

if %scan('{#':dta) = 1
and %scan('}':dta) > 0;
    component = 'icetag';
endif;

if %scan('(#':dta) = 1
and %scan(')':dta) > 0;
    component = 'icetag';
endif;

if %scan('('*:dta) = 1
and %scan(')':dta) > 0;
    component = 'icetag';
endif;

if %scan('('':dta) = 1
and %scan('/':dta) > 0
and %scan(')':dta) > 0;
    component = 'icetag';
endif;

if %scan('('':dta) = 1
and %scan(')':dta) > 0;
    component = 'icetag';
endif;

vtLocation = vtLocate(
    pVts
    :VT_INPUT
    :vtLinCol(lin:col)
    :VT_WEST
    :dta
);

// If field located Handle Tag
if vtLocation > *loval;

    vtGetField(pVts: vtLocation :vtField);

    if vtField.DSPATTR = FIELD_IS_HIDDEN;
        // setHidden(pVts: vtLocation);

        // WE NEED TO ITER ONLY IF WE DONT HAVE A FIELD WHERE THERE IS META DATA
    
```

```
// SINCE WE MIGHT NEED TO UPDATE A COMBOBOX OR RADIOGROUP
if last_field.metadata = '';
    pos = VT_NEXT;
    iter;
endif;
else;

    if vtField.metadata = '';

        if component = 'icetag';

            removeText = setupIceTag(
                pVts
                :vtLocation
                :dta
                :col
            );
        else;

            removeText = setupScreenComponent(
                pVts
                :vtLocation
                :dta
                :lin
                :col
            );

            ibmi = isIbmiField(
                dta
            );

        endif;
    else;

        if component <> '';
            removeText = *on;
        else;
            removeText = isScreenComponent(
                pVts
                :vtLocation
                :dta
                :lin
                :col
            );

            ibmi = isIbmiField(
                dta
            );

        endif;
    endif;
endif;
```

```
endif;

vtGetField(pVts: vtLocation :vtField);

last_field = vtField;
last_loc = vtLocation;
endif;

endif;

// If field changed then remove text accordingly
if removeText = *ON;

if ibmi = *off;
    // emLSetSkipOutputPosition(
    // lin
    // :lin
    // :col
    // :col+len
    // );

    vtWinPatch(pVts: lin: col: ': len);
else;

    offset = 0;
    start_col = col;
    end_col = col + len;

    // EX: 1-4, *SAME
    if %scan('*': dta) > 0
    and %scan(',': dta) > 0;
        offset = %scan(',': dta);
    endif;

    if offset > 0;
        offset = (offset - 1);
        start_col = col + offset;
        //end_col = %Len(%subst(dta: offset: %Len(dta) - offset));
        pad_len = len - offset;

        vtWinPatch(pVts: lin: start_col: ': pad_len);
    else;
        // emLSetSkipOutputPosition(
        // lin
        // :lin
        // :col
        // :col + len
        // );
        vtWinPatch(pVts: lin: col: ': len);
    endif;
endif;
```

```
endif;

endif;

// If field not handled - Check for Leading Text
else;
    chars = ' .:?'';
    wDta = %trimr(dta: chars);

    vtLocation = vtLocate(
        pVts
        :VT_INPUT
        :vtLinCol(lin:col)
        :VT_EAST
        :wDta
    );

    if vtLocation > *loval;

        vtGetField(pVts: vtLocation :vtField);

        if vtField.DSPATTR = FIELD_IS_HIDDEN;
            // setHidden(pVts: vtLocation);
            pos = VT_NEXT;
            iter;
        endif;

        if vtField.metadata = '';

            removeText = setupIceTag(
                pVts
                :vtLocation
                :wDta
                :col
            );

            if removeText = *off;

                removeText = setupScreenComponent(
                    pVts
                    :vtLocation
                    :wDta
                    :lin
                    :col
                );

                ibmi = isIbmiField(
                    dta
                );
```

```
        endif;

    endif;

    vtGetField(pVts: vtLocation :vtField);

    last_field = vtField;
    last_loc = vtLocation;

endif;

endif;

if removeText = *off
and dta <> ''
and last_field.metadata <> '';

    if attr <> FIELD_IS_HIDDEN;

        removeText = updateScreenComponent(
            pVts
            :last_loc
            :last_field
            :dta
        );

        vtGetField(pVts: last_loc :last_field);

        if (removeText);

            vtWinPatch(pVts: lin :col : '' : len);

            // emlSetSkipOutputPosition(
            //   lin:lin:col:col+len
            // );

        endif;

    endif;

endif;

pos = VT_NEXT;
enddo;

return;
```

```

end-proc;

/* ----- */
  Action field
/* ----- */
dcl-proc setupActionField;

    dcl-pi *n          ind;
        pVts          pointer value;
        vtLocation    liked(vtLocationDS) value;
    end-pi;

    dcl-s  c          int(10);
    dcl-s  cnt        int(10);
    dcl-s  maxLen     int(10);
    dcl-s  pMeta      pointer;
    dcl-ds vtField    liked(vtFieldDS);

    vtGetField(pVts: vtLocation :vtField);

    // metadata = xlateStr(`{
    //   "xtype":"ACTIONFIELD",
    //   "value": "${vtField.value}"
    // }`: 277:0);

    pMeta = json_newObject();
    json_setStr(pMeta:'xtype':'ACTIONFIELD');
    json_setStr(pMeta:'value':vtField.value);

    vtSetMetaData(
        pVts
        :vtLocation
        :JSON_ASJSONTEXT16M(pMeta)
    );

    json_delete(pMeta);

    return *ON;

end-proc;

/* ----- */
  DropDown field
  - output field with ", "
  - ex. PPD, RTL, INV, ALT,
/* ----- */
dcl-proc setupComboboxField;

    dcl-pi *n          ind;
        pVts          pointer value;

```

```

        vtLocation      likeds(vtLocationDS) value;
        dta              varchar(256) value;
        ibmi             ind value;
    end-pi;

    dcl-s pValues         pointer;
    dcl-s pValue          pointer;
    dcl-s value           varchar(132) dim(99);
    dcl-s c               int(10);
    dcl-s index           int(10);
    dcl-s cnt             int(10);
    dcl-s flex            int(10);
    dcl-ds vtField        likeds(vtFieldDS);
    dcl-s removeText      ind;
    dcl-s temp            varchar(32);
    dcl-s emptyText       varchar(32);

    dcl-s pMeta           pointer;

    removeText = *OFF;

    // create array with values and save max length for values
    value = '';
    index = 0;

    cnt = words(%trimr(dta: ',' ) :',' );

    vtGetField(pVts: vtLocation :vtField);

    for c = 1 to cnt;

        // THIS IS PRIMARY FOR IBM I SYSTEM SCREENS
        temp = %trim(word(dta :c :',' ));

        if %len(temp) <= vtField.maxLen
            and %scan('...': temp) = 0;

            if ibmi;

                if %subst(temp: 1: 1) = '*';

                    index += 1;
                    value(index) = temp;

                else;
                    if emptyText = '';
                        emptyText = temp;
                    endif;
                endif;
            else;

```

```
        // USED FOR NONE STANDARD SCREENS
        index += 1;
        value(index) = temp;
    endif;
endif;

endfor;

// Compare field length with max length for values

if cnt > 0;

    if index < cnt;
        cnt = index;
    endif;

    pValues = json_newArray();
    for c = 1 to cnt;
        pValue = json_newObject();
        json_setstr(
            pValue
            : 'key'
            : value(c)
        );
        json_setstr(
            pValue
            : 'value'
            : value(c)
        );
        json_arrayPush(pValues :pValue);
    endfor;

    if vtField.maxLen <= 6;
        flex = vtField.maxLen + 3;
    else;
        flex = vtField.maxLen + 2;
    endif;

    pMeta = json_newObject();
    json_setStr(pMeta: 'xtype': 'COMBOBOX');
    json_setStr(pMeta: 'value': vtField.value);
    json_setInt(pMeta: 'flex': flex);
    json_moveObjectInto(pMeta: 'valueMap': pValues);

    if emptyText <> '';
        json_setStr(pMeta: 'emptyText': emptyText);
    endif;
end
```

```

        vtSetMetaData(
            pVts
            :vtLocation
            :JSON_ASJSONTEXT16M(pMeta)
        );

        removeText = *ON;

    endif;

    return removeText;

end-proc;

/* ----- */
Radio field
- output field with both , and =
- ex. 1=Royalty, 2=Unit Rate, 3=Receipts
/* ----- */
dcl-proc setupRadioButtonField;

    dcl-pi *n                ind;
        pVts                pointer value;
        vtLocation          likeds(vtLocationDS) value;
        dta                  varchar(256) value;
        flex                 int(5);

    end-pi;

    dcl-s pValues            pointer;
    dcl-s pValue             pointer;
    dcl-s work               varchar(132);
    dcl-s key                varchar(132) dim(99);
    dcl-s temp               varchar(132);
    dcl-s button_name        varchar(132);
    dcl-s value              varchar(132) dim(99);
    dcl-s c                  int(10);
    dcl-s cnt                int(10);
    dcl-s maxlen             int(10);
    dcl-ds vtField           likeds(vtFieldDS);
    dcl-s removeText         ind;
    dcl-s index              int(10);
    dcl-s len                int(10);
    dcl-s pMeta              pointer;
    dcl-s defaults           pointer;

    removeText = *OFF;

    // create array with values and save max length for values
    key = '';

```

```

value = '';
maxLen = 0;
cnt = words(%trimr(dta: ',' :','));
for c = 1 to cnt;
    work = word(dta :c :',' );
    key(c) = %trim(word(work :1 :'='));
    value(c) = %trim(word(work :2 :'='));
    len = %len(%trim(key(c)));

    // NEED TO TO CHECK FOR BLANK OPTIONS

    if %scan(key(c): `` ``) = 1;
        key(c) = '';
        len = 0;
    endif;

    if (len > maxLen);
        maxLen = len;
    endif;
endfor;

// Compare field length with max length for values
vtGetField(pVts: vtLocation :vtField);
if (vtField.maxLen >= maxLen);

    removeText = *ON;
    index = findFieldIndex(pVts: vtLocation);

    button_name = 'rb-' +
                  %trim(%char(vtField.lin)) +
                  '-' +
                  %trim(%char(vtField.col));

    pValues = json_newArray();
    for c = 1 to cnt;
        pValue = json_newObject();
        json_setstr(
            pValue
            : 'inputValue'
            : key(c)
        );
        json_setstr(
            pValue
            : 'boxLabel'
            : value(c)
        );
        json_setstr(
            pValue
            : 'name'
            : button_name

```

```

    );
    json_setBool(
        pValue
        : 'isFormField'
        : *OFF
    );
    json_setInt(
        pValue
        : 'tabIndex'
        : index
    );

    if key(c) = '' and vtField.value = '';
        json_setBool(
            pValue
            : 'checked'
            : *on
        );
    endif;

    json_arrayPush(pValues :pValue);
endfor;

defaults = json_newObject();
json_setStr(defaults: 'margin': '0 20 0 0');

pMeta = json_newObject();
json_setStr(pMeta: 'xtype': 'radiogroup');
json_setStr(pMeta: 'value': vtField.value);
json_setBool(pMeta: 'simpleValue': *ON);
json_setInt(pMeta: 'flex': flex);
json_moveObjectInto(pMeta: 'items': pValues);
json_moveObjectInto(pMeta: 'defaults': defaults);

vtSetMetaData(
    pVts
    : vtLocation
    : JSON_ASJSONTEXT16M(pMeta)
);

json_delete(pMeta);

endif;

return removeText;

end-proc;

/* ----- */
Find line number for string

```

```
\* ----- */
dcl-proc findLineNumber;

    dcl-pi *n            int(10);
        pVts            pointer value;
        search          varchar(256) value;
    end-pi;

    dcl-ds vtLocation    likeds(vtLocationDS);

    vtLocation = vtLocate(
        pVts
        :VT_OUTPUT
        :vtLinCol(1:1)
        :VT_AT
        :search
    );

    return vtLocation.lin;

end-proc;

/* ----- */
Set hidden on field
/* ----- */
dcl-proc setHidden;

    dcl-pi *n;
        pVts            pointer;
        vtLocation      likeds(vtLocationDS);
    end-pi;

    dcl-s pMeta          pointer;

    pMeta = json_newObject();

    json_setBool(pMeta: 'hidden': *on);

    vtSetMetaData(
        pVts
        :vtLocation
        :JSON_ASJSONTEXT16M(pMeta)
    );

    json_delete(pMeta);

end-proc;

/* ----- */
```

```

    Get next data from list
\* ----- */
dcl-proc getNextDta;

    dcl-pi *n                varchar(256);
        pVts                pointer;
        pos                 int(5);
        dta                 varchar(256);
    end-pi;

    dcl-s len                int(5);
    dcl-s col                int(5);
    dcl-s lin                int(5);
    dcl-s off                int(5);
    dcl-s attr               char(1);
    dcl-s dtaNext            varchar(256);
    dcl-s dtaTrim            varchar(256);
    dcl-ds vtLocation        likeds(vtLocationDS);

    dtaTrim = dta;

    vtGetOutputList(pVts:pos:lin:col:off:len:attr:dtaNext);

    vtLocation = vtLocate(
        pVts
        :VT_INPUT
        :vtLinCol(lin:col)
        :VT_WEST
        :dta
    );

    if vtLocation = *loval;

        vtLocation = vtLocate(
            pVts
            :VT_INPUT
            :vtLinCol(lin:col)
            :VT_EAST
            :dta
        );

    if vtLocation = *loval;
        if (%subst(dta :%len(%trim(dta)) :1) = ',');
            pos = %scan('= ' :dta);
            if pos = 0;

                dtaNext = %trim(dtaTrim) + ' ' + %trim(dtaNext) + ',';

                // emlSetSkipOutputPosition(
                //   lin:lin:col:col+len

```

```
        // );

        endif;
    endif;
else;
    dtaNext = dta;
endif;
else;
    dtaNext = dta;
endif;

return dtaNext;

end-proc;

/* ----- */
Set subfile config option (integer)
/* ----- */
dcl-proc setCfgInt;

    dcl-pi *n;
        config          varchar(32768);
        option          varchar(256) value;
        value           int(10) value;
    end-pi;

    strAppend(config :',' :option + ':' + %char(value));

    return;

end-proc;

/* ----- */
Set subfile config option (string)
/* ----- */
dcl-proc setCfgStr;

    dcl-pi *n;
        config          varchar(32768);
        option          varchar(256) value;
        value           varchar(256) value;
    end-pi;

    strAppend(config :',' :option + ':' + value + '');

    return;

end-proc;

/* ----- */
```



```
        config,
        tooltip,
        emptytext
    from ICETAG
    WHERE ID = '${dta}' and active = 1
`);

pSql = json_sqlResultRow(sqlStmt);

pConfig = json_locate(pSql: 'config');

if *inu1 = *on;
    json_text = JSON_ASJSONTEXT16M(pSql);
    json_text = JSON_ASJSONTEXT16M(pConfig);
endif;

emptytext = json_getStr(pSql: 'emptytext');
tooltip = json_getStr(pSql: 'tooltip');

json_setStr(pConfig: 'value': vtField.value);

if emptytext <> '';
    json_setStr(pConfig: 'emptyText': emptytext);
endif;

if tooltip <> '';
    json_setStr(pConfig: 'tooltip': tooltip);
endif;

pMeta = createIceTag(pVts: vtLocation: pConfig: vtField: dta: dtaCol);

if pMeta <> *null;

    if *inu1 = *on;
        json_text = JSON_ASJSONTEXT16M(pMeta);
    endif;

    removeText = *ON;

    vtSetMetaData(
        pVts
        :vtLocation
        :JSON_ASJSONTEXT16M(pMeta)
    );

    json_delete(pMeta);
    json_delete(pSql);

endif;
```

```

        return removeText;

end-proc;

/* ----- */
DropDown field
- output field with ","
- ex. PPD, RTL, INV, ALT
/* ----- */
dcl-proc createIceTag;

    dcl-pi *n                pointer;
        pVts                pointer value;
        vtLocation          likeds(vtLocationDS) value;
        config              pointer value;
        vtField             likeds(vtFieldDS);
        dta                 varchar(256);
        dtaCol              int(5);
    end-pi;

    dcl-s pMeta              pointer;
    dcl-s type               varchar(32);
    dcl-s index              int(10);
    dcl-s width              int(5);
    dcl-s flex               int(5);
    dcl-s emptytext          varchar(256);
    dcl-s tooltip            varchar(2048);
    dcl-s fieldType          int(5);
    dcl-s isEditable         ind;

    // NEED TO FIGURE OUT WHAT TO DO
    fieldType = vtFfwDataType(vtField);

    // BuildComponent
    type = json_getStr(config:'type');

    width = json_getInt(config: 'width': 0);
    emptyText = json_getStr(config: 'emptyText');
    tooltip = json_getStr(config: 'tooltip');

    flex = json_getInt(config: 'flex': 0);

    if flex = 0;

        if dtaCol < vtField.col;
            flex = vtField.maxLen;
        else;

```

```
        flex = ((dtaCol - vtField.col) + %len(dta));
    endif;
    flex = %dec ((flex / 5): 5: 0) * 5;

    if flex < vtField.maxLen;
        flex = vtField.maxLen;
    endif;
endif;

select;
when    type = 'DDSQL';
    isEditable = *ON;
    pMeta = create_ddsql(config: vtLocation: pVts);
when    type = 'datefield';
    flex = 0;
    isEditable = *ON;
    pMeta = create_datefield(config);
when    type = 'checkboxfield';
    pMeta = create_checkboxfield(config);
when    type = 'radiogroup';
    index = findFieldIndex(pVts: vtLocation);
    pMeta = create_radiogroup(config: index);
when    type = 'combobox';
    isEditable = *ON;
    pMeta = create_combobox(config);
when    type = 'actionfield';
    flex = 0;
    pMeta = create_actionfield(config);
when    type = 'textfield';
    pMeta = create_textfield(config);
    flex = 0;
when    type = 'numberfield';
    pMeta = create_numberfield(config);
    flex = 0;
when    type = 'websearch';
    pMeta = create_websearch(config);
when    type = 'mapfield';
    pMeta = create_mapfield(config);
when    type = 'functionfield';
    pMeta = create_functionfield(config);
endsl;

if width <> 0;
    json_setInt(pMeta: 'width': width);
endif;

if flex <> 0;
    json_setInt(pMeta: 'flex': flex);
```

```

else;
  if type = 'actionfield';
    if vtField.maxLen <= 6;
      json_setInt(pMeta: 'flex': vtField.maxLen + 3);
    else;
      json_setInt(pMeta: 'flex': vtField.maxLen + 2);
    endif;
  endif;
endif;

if emptyText <> '';
  json_setStr(pMeta: 'emptyText': emptyText);
endif;

if tooltip <> '';
  json_setStr(pMeta: 'tooltip': tooltip);
endif;

if isEditable;
  if fieldType = VT_FFW_NUMERIC_SHIFT
  or fieldType = VT_FFW_DIGITS_ONLY
  or fieldType = VT_FFW_NUMERIC_ONLY;
    json_setStr(pMeta: 'ffw': '');
  endif;
endif;

if vtFfwIsProtected(vtField);
  json_setBool(pMeta: 'readOnly': *ON);
endif;

return pMeta;

end-proc;

/* ----- */
DropDown field
- output field with ","
- ex. PPD, RTL, INV, ALT
/* ----- */
dcl-proc create_ddsql;

  dcl-pi *n          pointer;
    config          pointer value;
    vtLocation      liked(vtLocationDS) value;
    pVts            pointer value;
  end-pi;

  dcl-s pMeta        pointer;
  dcl-s pTextColumns pointer;
  dcl-s pListColumns pointer;

```

```
dcl-s pSearch          pointer;
dcl-s sqlstmt          varchar(8192);
dcl-s pSql            pointer;
dcl-ds itTextColumns   likeds(json_iterator);
dcl-s asText          varchar(512);
dcl-s asValue         varchar(128);
dcl-s asFile          varchar(128);
dcl-s asLibrary       varchar(128);
dcl-s asWhere         varchar(512);
dcl-s df             varchar(128);
dcl-s dc             varchar(128);
dcl-s dt             varchar(128);
dcl-s dpAsRaw        ind;
dcl-s valueAsRaw     ind;
dcl-s popup          ind;
dcl-s hideTrigger    ind;
dcl-s selectOnTab    ind;
dcl-s splitSearch    ind;
dcl-s dependOnNext   ind;
dcl-s counter        int(5);
dcl-s width          int(5);
dcl-s flex           int(5);
dcl-ds emlField      likeds(emlFieldDS);

pTextColumns   = json_locate(config : 'textColumns');
pListColumns   = json_locate(config : 'displayList');

itTextColumns = json_setIterator(pTextColumns: '');

counter = 0;

dow json_forEach(itTextColumns);

    if asText = '';
        asText = json_getValueStr(itTextColumns.this);
        counter +=1;
    else;
        if counter = 1;
            asText = %trim(asText) +
                ' CONCAT '' - '' CONCAT ' + json_getValueStr(itTextColumns.this);
        else;
            asText = %trim(asText) +
                ' CONCAT '' ' CONCAT ' + json_getValueStr(itTextColumns.this);
        endif;
        counter +=1;
    endif;

enddo;
```

```

asFile = json_getStr(config: 'fileName');
asLibrary = json_getStr(config: 'library');
asWhere = json_getStr(config: 'where');
df = json_getStr(config: 'dependencyField: ');
dc = json_getStr(config: 'dependencyColumn: ');
dt = json_getStr(config: 'dependencyType: 'string');
asValue = json_getStr(config: 'valueColumn');
dpAsRaw = json_getBool(config: 'dependencyAsRaw');
valueAsRaw = json_getBool(config: 'asRaw');
popup = json_getBool(config: 'popup');
hideTrigger = json_getBool(config: 'hideTrigger');
selectOnTab = json_getBool(config: 'selectOnTab');
selectOnTab = json_getBool(config: 'selectOnTab');
splitSearch = json_getBool(config: 'splitSearch');
dependOnNext = json_getBool(config: 'dependOnNext');

if popup;
    hideTrigger = *on;
endif;

if asLibrary <> '*LIBL';
    asFile = %trim(asLibrary) + '.' + asFile;
endif;

if asWhere <> '';
    asWhere = 'where ' + %trim(asWhere);
endif;

if df <> '' and dc <> '';
    if asWhere = '';
        if dt = 'string';
            asWhere = 'where lower(' + dc + ') in ( ' +
                'case when ':dependency' <> '''' then ' +
                'lower('':dependency') ' +
                'else lower(' + dc + ') ' +
                'end ' +
                ')';
        else;
            asWhere = 'where ' + dc + ' in ' +
                '(' +
                ' case when :dependency <> '''' ' +
                ' then :dependency ' +
                ' else ' + dc + ' ' +
                ' end ' +
                ')';
        endif;
    else;
        if dt = 'string';
            asWhere =
                %trim(asWhere) + ' and lower(' + dc + ') in ' +

```

```

        ' ' +
        ' case when '' :dependency'' <> '' '' ' +
        ' then lower('' :dependency'') ' +
        ' else lower(' + dc + ' ) ' +
        ' end ' +
        ' )';
    else;
        asWhere =
            %trim(asWhere) + ' and ' + dc + ' in ' +
            ' ( ' +
            ' case when :dependency <> '' '' ' +
            ' then :dependency ' +
            ' else ' + dc + ' ' +
            ' end ' +
            ' )';
    endif;
endif;
endif;

sqlStmt = (
    with rows as (
        select
            ${asText} as "text",
            ${asValue} as "value"
        from ${asFile}
        ${asWhere}
    )
);

sqlStmt += (
    select *
    from rows
    :where
    order by "text"
    limit :limit offset :start
);

sqlStmt = xlateStr(sqlStmt: 277:0);

pSql = json_parseString(extSql('{}' :sqlStmt));

// BuildComponent

pMeta = json_newObject();
json_setStr(pMeta:'xtype':'DDSQL');
json_setint(pMeta : 'maxLength': 90);
json_setstr(pMeta : 'sqlStmt': sqlStmt);
json_setbool(pMeta : 'forceSelection': *off);
json_setbool(pMeta : 'asRaw': valueAsRaw);
json_setbool(pMeta : 'anyMatch': *on);

```

```

json_setstr(pMeta : 'maskRe': '');
json_setBool(pMeta: 'selectOnTab': selectOnTab);
json_setbool(pMeta : 'hideTrigger': hideTrigger);
json_setbool(pMeta : 'matchFieldWidth': *off);
json_setStr(pMeta : 'valueField': 'value');
json_setbool(pMeta : 'splitSearch': *on);
json_setStr(pMeta : 'displayField': 'text');

json_setbool(pMeta: 'popup': popup);
json_setbool(pMeta: 'dependencyAsRaw': dpAsRaw);

pSearch = json_locate(config: 'splitSearch');

if pSearch <> *null;
    json_setbool(pMeta : 'splitSearch' : splitSearch);
endif;

if pListColumns <> *null;
    json_moveObjectInto(pMeta: 'displayList': pListColumns);
endif;

if df <> '';
    json_setStr(pMeta: 'dependency': df);
endif;

if dependOnNext;
    emlField = emlGetField(pVts :vtLocation);
    json_setstr(pMeta: 'dependency': 'i' + %char(emlField.fieldNo));
endif;

json_copyValue(pMeta: 'sqlVoucher': pSql: 'sqlVoucher');

return pMeta;

end-proc;

/* ----- */
Date field
/* ----- */
dcl-proc create_datefield;

    dcl-pi *n                pointer;
        config              pointer value;
    end-pi;

    dcl-s dateFormat        varchar(20);
    dcl-s altFormat          varchar(256);
    dcl-s pMeta              pointer;

```

```
dateFormat = json_getStr(config: 'format');

pMeta = json_newObject();
json_setStr(pMeta:'xtype':'datefield');
json_setStr(pMeta : 'format' : dateFormat);
json_setStr(pMeta : 'ffw' : '');
json_setstr(pMeta : 'altFormats' : `Y-m-d-H.i.s.u
|d-n-Y
|d-n-y
|j-n-Y
|j-n-y
|d/n/Y
|d/n/y
|j/n/Y
|j/n/y
|d-m-Y
|d-m-y
|j-m-Y
|j-m-y
|d/m/Y
|d/m/y
|j/m/Y
|j/m/y
|m/d/Y
|m/d/y
|m/j/Y
|m/j/y
|m-d-Y
|m-d-y
|m/j/Y
|m/j/y
|n/d/Y
|n/d/y
|n/j/Y
|n/j/y
|n-d-Y
|n-d-y
|n-j-Y
|n-j-y
|Y-m-d
|y-m-d
|Y-n-d
|y-n-d
|Y-m-j
|y-m-j
|Y-n-j
|y-n-j
|Y/m/d
|y/m/d
|Y/n/d
```

|y/n/d
|Y/m/j
|y/m/j
|Y/n/j
|y/n/j
|d.m.Y
|d.m.y
|j.m.Y
|j.m.y
|d.n.Y
|d.n.y
|j.n.Y
|j.n.y
|m.d.Y
|m.d.Y
|m.j.Y
|m.j.y
|n.d.Y
|n.d.y
|n.j.Y
|n.j.y
|Y.m.d
|y.m.d
|Y.m.j
|y.m.j
|Y.n.d
|y.n.d
|Y.n.j
|y.n.j
|dmY
|dmy
|jmY
|jmy
|dnY
|dny
|jnY
|jny
|mdY
|mdy
|mjY
|m jy
|ndY
|ndy
|njY
|njy
|Ymd
|ymd
|Ymj
|ymj
|Ynd

```

                                |ynd
                                |Ynj
                                |ynj|`
);

    return pMeta;

end-proc;

/* ----- */
Checkbox field
/* ----- */
dcl-proc create_checkboxfield;

    dcl-pi *n                pointer;
           config            pointer value;
    end-pi;

    dcl-s pMeta              pointer;
    dcl-s trueValue          varchar(5);
    dcl-s falseValue        varchar(5);
    dcl-s fieldValue         varchar(10);

    trueValue = json_getStr(config: 'trueValue');
    falseValue = json_getStr(config: 'falseValue');
    fieldValue = json_getStr(config: 'value');

    pMeta = json_newObject();
    json_setstr(pMeta:'xtype':'checkbox');
    json_setstr(pMeta:'value': fieldValue);
    json_setstr(pMeta:'trueValue':trueValue);
    json_setstr(pMeta:'falseValue':falseValue);

    return pMeta;

end-proc;

/* ----- */
Radiogroup field
/* ----- */
dcl-proc create_radiogroup;

    dcl-pi *n                pointer;
           config            pointer value;
           index             int(10);
    end-pi;

    dcl-s pMeta              pointer;
    dcl-s fieldValue         varchar(10);
```

```

dcl-s items          pointer;
dcl-s defaults       pointer;
dcl-ds itItems       likeds(json_iterator);
dcl-s itemValue       varchar(10);

fieldValue = json_getStr(config: 'value');
items = json_locate(config: 'items');

defaults = json_newObject();
json_setStr(defaults: 'margin': '0 20 0 0');

itItems = json_setIterator(items: '');

dow json_forEach(itItems);

    json_setBool(itItems.this: 'isFormField': *OFF);
    json_setInt(itItems.this: 'tabIndex': index);
enddo;

pMeta = json_newObject();
json_setstr(pMeta: 'xtype': 'radiogroup');
json_setstr(pMeta: 'value': fieldValue);
json_setbool(pMeta: 'simpleValue': *ON);
json_moveObjectInto(pMeta: 'items': items);
json_moveObjectInto(pMeta: 'defaults': defaults);

return pMeta;

end-proc;

/* ----- */
/* Combobox field */
/* ----- */
dcl-proc create_combobox;

    dcl-pi *n          pointer;
        config        pointer value;
    end-pi;

    dcl-s pMeta        pointer;
    dcl-s fieldValue    varchar(256);
    dcl-s fields        pointer;
    dcl-s store         pointer;
    dcl-s data          pointer;
    dcl-s field         pointer;
    dcl-s style         pointer;
    dcl-s ffw           varchar(32);

    fieldValue = json_getStr(config: 'value');
    // ffw = json_getStr(config: 'ffw: ');

```

```

data = json_locate(config: 'values');
style = json_locate(config: 'fieldStyle');

fields = json_newArray();
field = json_newObject();

json_setStr(field: 'name': 'text');
json_setStr(field: 'type': 'auto');

json_arrayPush(fields: field);

field = json_newObject();

json_setStr(field: 'name': 'value');
json_setStr(field: 'type': 'auto');

json_arrayPush(fields: field);

store = json_newObject();
json_setbool(store: 'autoLoad': *ON);
json_moveObjectInto(store: 'fields': fields);
json_moveObjectInto(store: 'data': data);

pMeta = json_newObject();
json_setstr(pMeta: 'xtype': 'combobox');
json_setstr(pMeta: 'value': %trim(fieldValue));
json_setstr(pMeta: 'displayField': 'text');
json_setstr(pMeta: 'valueField': 'value');
json_setBool(pMeta: 'matchFieldWidth': *OFF);
json_setInt(pMeta: 'maxLength': 999);
json_setInt(pMeta: 'minChars': 1);
json_setStr(pMeta: 'queryMode': 'local');
json_setbool(pMeta: 'remoteFilter' :*OFF);
json_setbool(pMeta: 'forceAll' :*OFF);
json_setbool(pMeta: 'remoteSort' :*OFF);
json_setbool(pMeta: 'queryCaching' :*OFF);
json_setbool(pMeta: 'anyMatch' :*ON);
// json_setStr(pMeta: 'ffw': ffw);
json_moveObjectInto(pMeta: 'store': store);

if style <> *null;
    json_moveObjectInto(pMeta: 'fieldStyle': style);
endif;

return pMeta;

end-proc;

/* ----- */
Action field

```

```
\* ----- */
dcl-proc create_actionfield;

    dcl-pi *n            pointer;
        config          pointer value;
    end-pi;

    dcl-s pMeta          pointer;
    dcl-s fieldValue     varchar(256);
    dcl-s command        varchar(10);
    dcl-s width          int(5);
    dcl-s flex           int(5);

    fieldValue = json_getStr(config: 'value');
    command = json_getStr(config: 'command');

    pMeta = json_newObject();
    json_setStr(pMeta:'xtype':'ACTIONFIELD');
    json_setStr(pMeta:'value':fieldValue);

    if command <> '';
        json_setStr(pMeta: 'cmd': command);
    endif;

    return pMeta;

end-proc;

/* ----- */
textfield
/* ----- */
dcl-proc create_textfield;

    dcl-pi *n            pointer;
        config          pointer value;
    end-pi;

    dcl-s pMeta          pointer;

    pMeta = json_newObject();
    json_setStr(pMeta:'xtype':'textfield');

    return pMeta;

end-proc;

/* ----- */
numberfield
/* ----- */
dcl-proc create_numberfield;
```

```
dcl-pi *n          pointer;
      config       pointer value;
end-pi;

dcl-s pMeta        pointer;

pMeta = json_newObject();
json_setStr(pMeta:'xtype':'numberfield');

return pMeta;

end-proc;

/* ----- */
websearch field
/* ----- */
dcl-proc create_websearch;

dcl-pi *n          pointer;
      config       pointer value;
end-pi;

dcl-s pMeta        pointer;
dcl-s url          varchar(256);

url = json_getStr(config: 'searchUrl': '');

pMeta = json_newObject();
json_setStr(pMeta:'xtype':'icecap-field-websearch');
json_setStr(pMeta: 'searchUrl': url);

return pMeta;

end-proc;

/* ----- */
Google map field
/* ----- */
dcl-proc create_mapfield;

dcl-pi *n          pointer;
      config       pointer value;
end-pi;

dcl-s pMeta        pointer;
dcl-s url          varchar(256);

pMeta = json_newObject();
```

```

        json_setStr(pMeta:'xtype':'icecap-field-map');

        return pMeta;

end-proc;

/* ----- */
Function field
/* ----- */
dcl-proc create_functionfield;

    dcl-pi *n                pointer;
           config            pointer value;
    end-pi;

    dcl-s pMeta              pointer;
    dcl-s url                varchar(256);

    pMeta = json_newObject();
    json_setStr(pMeta:'xtype':'icecap-field-function');
    json_setStr(pMeta:'func': json_getStr(config: 'func': ''));
    json_setStr(pMeta:'initFunc': json_getStr(config: 'initFunc': ''));
    json_setBool(pMeta: 'fixedSize': json_getBool(config: 'fixedSize'));

    return pMeta;

end-proc;

/* ----- */
DropDown field
- output field with ","
- ex. PPD, RTL, INV, ALT
/* ----- */
dcl-proc setupScreenComponent;

    dcl-pi *n                ind;
           pVts              pointer;
           vtLocation         likeds(vtLocationDS);
           dta                varchar(256);
           lin                int(5);
           col                int(5);
    end-pi;

    dcl-s component          varchar(32);
    dcl-s removeText         ind;
    dcl-s pos                int(5);
    dcl-s index              int(5);
    dcl-s flex               int(5);

```

```

dcl-s dtalen          int(5);
dcl-ds vtField        likeds(vtFieldDS);
dcl-s pConfig         pointer;
dcl-s pMeta           pointer;
dcl-s trueValue       varchar(5);
dcl-s falseValue      varchar(5);
dcl-c f4_1            'F4';
dcl-c f4_2            'F4=';
dcl-s f4_1Value       varchar(132);
dcl-s f4_2Value       varchar(132);
dcl-s dtaValue        varchar(132);
dcl-s ibmi            ind;

dta = %trim(dta);

f4_1Value = LowerCase(f4_1);
f4_2Value = LowerCase(f4_2);
dtaValue = LowerCase(dta);

removeText = *OFF;
ibmi = *OFF;

select;
when
    %scan(f4_2Value: dtaValue) > 0
    or %scan(f4_1Value: dtaValue) > 0;

    if %len(dtaValue) >=2;
        if %len(dtaValue) > 2;

            // if %subst(dtaValue: 3: 1) = ' '
            // or %subst(dtaValue: 3: 1) = '=';
            // component = 'actionfield';
            // endif;
            component = 'actionfield';

        else;
            component = 'actionfield';
        endif;
    endif;
when
    %scan('*': dtaValue) > 0
    and %scan('...': dtaValue) > 0;
    component = 'actionfield';
when %scan('= ' :dta) > 0
and  dta <> '==>';
    // NEED TO CHECK THAT WE DONT CREATE BUTTONS IF THE
    // TEXT DOES NOT HAVE = AT POSITION 2 IN THE STRING
    // EX. Name, *NONE, F4=Prompt (Bad)
    // Ex. 4=Test, 5=Test2 (Good)

```

```

        if %subst(dta: 2: 1) = '='
        or %subst(dta: 4: 1) = '=';
            component = 'radiogroup';
        endif;
when words(%trimr(dta: ',') :',') > 1;
    if %scan('*': dta) > 0
    and %scan(':', dta) > 0;
        component = 'combobox';
        ibmi = *ON;
    else;
        if %subst(dta :%len(%trim(dta)) :1) = ',';
            component = 'combobox';
        endif;
    endif;
when %scan('/: dta) = 2
and %len(dta) = 3;
    component = 'checkbox';
endsl;

```

// Manipulate Component Per Type

```

if component > '';

    vtGetField(pVts: vtLocation :vtField);

    if col < vtField.col;
        flex = vtField.maxLen;
    else;
        flex = ((col - vtField.col) + %len(dta));
    endif;

    flex = %dec ((flex / 5): 5: 0) * 5;

    select;
    when component = 'combobox';

        removeText = setupComboBoxField(
            pVts
            :vtLocation
            :dta
            :ibmi
        );

    when component = 'radiogroup';
        removeText = setupRadioButtonField(
            pVts
            :vtLocation
            :dta
            :flex
        );
    when component = 'checkbox';

```

```
trueValue = %subst(dta: 1: 1);
falseValue = %subst(dta: 3: 1);

pConfig = json_newObject();
json_setStr(pConfig: 'type': 'checkboxfield');
json_setStr(pConfig: 'value': vtField.value);
json_setStr(pConfig: 'trueValue': trueValue);
json_setStr(pConfig: 'falseValue': falseValue);

pMeta = createIceTag(
    pVts:
    vtLocation:
    pConfig:
    vtField:
    dta:
    col
);

if pMeta <> *null;

    removeText = *ON;

    vtSetMetaData(
        pVts
        :vtLocation
        :JSON_ASJSONTEXT16M(pMeta)
    );

    json_delete(pMeta);

endif;
when component = 'actionfield';

pConfig = json_newObject();
json_setStr(pConfig: 'type': 'actionfield');
json_setStr(pConfig: 'value': vtField.value);

pMeta = createIceTag(pVts: vtLocation: pConfig: vtField: dta: col);

if pMeta <> *null;

    removeText = *OFF;

    if %scan(dtaValue: f4_1Value) = 1
    or %scan(dtaValue: f4_2Value) = 1;
        removeText = *ON;
    endif;

    vtSetMetaData(
```

```
        pVts
        :vtLocation
        :JSON_ASJSONTEXT16M(pMeta)
    );

    json_delete(pMeta);

endif;
endsl;
endif;

return removeText;

end-proc;

dcl-proc isScreenComponent;

    dcl-pi *n                ind;
        pVts                pointer;
        vtLocation          likeds(vtLocationDS);
        dta                  varchar(256);
        lin                  int(5);
        col                  int(5);
    end-pi;

    dcl-s component          varchar(32);
    dcl-s isComponent        ind;
    dcl-s pos                 int(5);
    dcl-s index               int(5);
    dcl-s flex                int(5);
    dcl-s dtalen              int(5);
    dcl-ds vtField            likeds(vtFieldDS);
    dcl-s pConfig             pointer;
    dcl-s pMeta               pointer;
    dcl-s trueValue           varchar(5);
    dcl-s falseValue         varchar(5);
    dcl-c f4_1                'F4';
    dcl-c f4_2                'F4=';
    dcl-s f4_1Value           varchar(132);
    dcl-s f4_2Value           varchar(132);
    dcl-s dtaValue            varchar(132);

    dta = %trim(dta);

    f4_1Value = LowerCase(f4_1);
    f4_2Value = LowerCase(f4_2);
    dtaValue = LowerCase(dta);

    select;
    when
```

```

%scan(f4_2Value: dtaValue) > 0
or %scan(f4_1Value: dtaValue) > 0;

if %len(dtaValue) >=2;
    if %len(dtaValue) > 2;

        if %subst(dtaValue: 3: 1) = ' '
            or %subst(dtaValue: 3: 1) = '=';
            component = 'actionfield';
        endif;

    else;
        component = 'actionfield';
    endif;
endif;
when
    %scan('*': dtaValue) > 0
    and %scan('...': dtaValue) > 0;
    component = 'actionfield';
when %scan('= ' :dta) > 0;
    // NEED TO CHECK THAT WE DONT CREATE BUTTONS IF THE
    // TEXT DOES NOT HAVE = AT POSITION 2 IN THE STRING
    // EX. Name, *NONE, F4=Prompt (Bad)
    // Ex. 4=Test, 5=Test2 (Good)
    if %subst(dta: 2: 1) = '='
    or %subst(dta: 4: 1) = '=';
        component = 'radiogroup';
    endif;
when words(%trimr(dta: ',') :',') > 1;
    if %scan('*': dta) > 0
    and %scan(',': dta) > 0;
        component = 'combobox';
    else;
        if %subst(dta :%len(%trim(dta)) :1) = ',';
            component = 'combobox';
        endif;
    endif;
when %scan('/': dta) = 2
and %len(dta) = 3;
    component = 'checkbox';
endsl;

if component <> '';
    isComponent = *on;
endif;

return isComponent;

end-proc;

```

```
dcl-proc isIbmiField;

    dcl-pi *n                ind;
        dta                  varchar(256);
    end-pi;

    dcl-s ibmi               ind;

    dta = %trim(dta);

    select;
    when words(%trimr(dta: ',') :',') > 1;
        if %scan('*': dta) > 0
            and %scan(':', dta) > 0;
            ibmi = *ON;
        endif;
    ends1;

    return ibmi;

end-proc;

dcl-proc updateScreenComponent;

    dcl-pi *n                ind;
        pVts                pointer;
        vtLocation           likeds(vtLocationDS);
        vtField              likeds(vtFieldDS);
        dta                  varchar(256);
    end-pi;

    dcl-s returnValue       ind;
    dcl-s pMeta              pointer;
    dcl-s pValueMap          pointer;
    dcl-s pValue             pointer;
    dcl-s cnt                int(10);
    dcl-s temp               varchar(32);
    dcl-s c                  int(10);
    dcl-s key                varchar(132);
    dcl-s value              varchar(132);
    dcl-s button_name        varchar(132);
    dcl-s index              int(10);
    dcl-s debugText          varchar(2048);

    dta = %trim(dta);

    // THIS ERROR PRONE WHEN DATA IS BLANK
    //if %subst(dta: 1: 1) = '*';

    // UPDATING COMBOBOX
```

```
if %scan('*': dta) = 1;

    pMeta = json_parseString(vtField.metadata);

    if pMeta <> *null;

        pValueMap = json_locate(pMeta: 'valueMap');

        if pValueMap <> *null;

            cnt = words(%trimr(dta: ',') :',');

            for c = 1 to cnt;

                temp = %trim(word(dta :c :','));

                if %scan('=': temp) = 0
                    and %scan('...': temp) = 0;

                    pValue = json_newObject();

                    json_setstr(
                        pValue
                        : 'key'
                        : temp
                    );

                    json_setstr(
                        pValue
                        : 'value'
                        : temp
                    );

                    json_arrayPush(pValueMap: pValue);
                endif;

            endfor;

            returnValue = *on;

            // debugText = JSON_ASJSOEXT16M(pMeta);

            vtSetMetaData(
                pVts
                : vtLocation
                : JSON_ASJSOEXT16M(pMeta)
            );

        endif;
    endif;
endif;
```

```
endif;

// UPDATE RADIOFIELD
if %scan('=': dta) > 1;

    pMeta = json_parseString(vtField.metadata);

    if pMeta <> *null;

        pValueMap = json_locate(pMeta: 'items');

        if pValueMap <> *null;

            index = findFieldIndex(pVts: vtLocation);

            button_name = 'rb-' +
                %trim(%char(vtField.lin)) +
                '-' +
                %trim(%char(vtField.col));

            cnt = words(%trimr(dta: ',') :',');

            for c = 1 to cnt;

                temp = %trim(word(dta :c :','));

                key = %trim(word(temp :1 :'='));
                value = %trim(word(temp :2 :'='));

                pValue = json_newObject();

                json_setstr(
                    pValue
                    : 'inputValue'
                    : key
                );
                json_setstr(
                    pValue
                    : 'boxLabel'
                    : value
                );
                json_setstr(
                    pValue
                    : 'name'
                    : button_name
                );
                json_setBool(
                    pValue
```

```
        : 'isFormField'
        : *OFF
    );
    json_setInt(
        pValue
        : 'tabIndex'
        : index
    );

    if key = '' and vtField.value = '';
        json_setBool(
            pValue
            : 'checked'
            : *on
        );
    endif;

    json_arrayPush(pValueMap: pValue);

endfor;

returnValue = *on;

// debugText = JSON_ASJSONTEXT16M(pMeta);

vtSetMetaData(
    pVts
    : vtLocation
    : JSON_ASJSONTEXT16M(pMeta)
);

endif;

endif;

endif;

json_delete(pValueMap);
json_delete(pMeta);

return returnValue;

end-proc;

dcl-proc findFieldIndex;

    dcl-pi *n                int(10);
        pVts                pointer value;
        vtLocation           liked(vtLocationDS) value;
    end-pi;
```

```
dcl-ds vtField          likeds(vtFieldDS);
dcl-ds vtFieldTemp      likeds(vtFieldDS);
dcl-ds vts              likeds(vtsDS) based(pVts);
dcl-s index             int(10);
dcl-s i                 int(10);

index = 9999;

vtGetField(pVts: vtLocation :vtField);

for i = 1 to vts.inputFields;
    vtWinGetFieldNo(pVts:i:vtFieldTemp);

    if vtField.lin = vtFieldTemp.lin
    and vtField.col = vtFieldTemp.col;
        index = (i -1);
    endif;
endfor;

return index;

end-proc;
```

Navigation

The RPG source code `navCapCorp.rpgle` demonstrates how easily a program call can be intercepted and a screen bypassed to immediately show the requested information, thus optimizing workflow.

This code originates from the Demo Environment and can be found in the folder:

`/iceapp/icecap2/plugins.`

```
<%@ language="RPGLE" dftactgrp="*NO" actgrp="*CALLER"%>
<%
ctl-opt copyright('System & Method (C), 2023');
ctl-opt decEdit('0,') datEdit(*YMD.);
ctl-opt debug(*YES);
ctl-opt bndDir('ICEBREAK':'ICEUTILITY':'ICECAPTERM':'ICECAPEMUL');

/*
----- */
/* Compile ....: CRTICEPGM STMF('/prj/iceCap/plugins/navcapcorp.rpgle') SVRID(PORTDEV)
LIBL(*SERVER) OBJLIB(ICECDEV) */
/* Compile ....: CRTICEPGM STMF('/prj/icecap/plugins/navcapcorp.rpgle') SVRID(PORTQA )
LIBL(*SERVER) OBJLIB(ICECQA) */
/* Compile ....: CRTICEPGM STMF('/prj/iceCap/plugins/navcapcorp.rpgle') SVRID(PORT761)
LIBL(*SERVER) OBJLIB(ICEC761) */
/* Compile ....: CRTICEPGM STMF('/prj/iceCap/plugins/navcapcorp.rpgle') SVRID(PORTQA )
LIBL(*SERVER) OBJLIB(ICECDIST) */
/*
----- */

/Include qAspHdr,ICEBREAK
/Include qAspHdr,ICEUTILITY
/include qasphdr,jsonparser
/include qrpgleref,icecapTerm
/include qrpgleref,icecapEmul

dcl-s pVts          pointer;
dcl-ds vts          likeds(vtsDS);
dcl-s wstr          varchar(500) ;
dcl-s wOption       varchar(500) ;
dcl-s wValue        varchar(500) ;

* -----
*/
c      *entry      plist
c      parm              vts
* -----
*/
/free
```

```
pVts = %addr(vts);
vts.showScreen = *off;

wStr = '/tmp/navcap-' + %char(%timestamp())
      + '.json';

Select;
  // Employees
  when vtGetScreen(pVts:1:2:10) = 'Corp. Data'
  and vtGetScreen(pVts:9:8:6) = 'EmpNo.';

  // Get employ id from paramenters
  wValue = json_getstr(vts.pParms : 'employid');

  // If we have a value perform action
  if wValue > '';
    wOption = json_getstr(vts.pParms : 'option': '5');
    vtSetFieldNo(pVts:2:wOption);
    vtSetFieldNo(pVts:3:wValue);
    vtPressKey(pVts: vtENTER);
  endif;

EndSl;

vts.showScreen = *on;

return;
```

Organization (PC Organizer)

The RPG source code `pcoServer.rpg1e` illustrates the additional processes performed when the `STRPCCMD` is called in the 5250 environment. This program can be customized to include extended security measures, additional business logic, changes to 5250 programs that cannot be recompiled, and codepage handling.

```

**free
  Ctl-Opt debug(*yes);
  Ctl-Opt actgrp(*caller);
  Ctl-Opt dftactgrp(*no);
  Ctl-Opt bnmdir('ICECAPTERM');
  Ctl-Opt bnmdir('ICEUTILITY');
* ----- */
* GO ICEBREAK */
* */
* CRTBNDRPG PGM(DEPLOYLIB/PCOSERVER) */
* SRCSTMF('/prj/icecap/plugins/pcoServer.rpg1e') */
* OPTION(*EVENTF) */
* DBGVIEW(*SOURCE) */
* TGTRLS(*CURRENT) */
* ----- */

/include qrpgleref,iceCapTerm
/include qasphdr,iceUtility

Dcl-S      cmdLow      char(512);
Dcl-S      URLPos      int(5) ;
Dcl-S      kolon      int(5) ;
Dcl-S      C          int(5) ;
Dcl-S      URLStart    int(5) ;
Dcl-S      URLEnd      int(5) ;
Dcl-S      URL         char(512);

dcl-pi *n;
  vts          likeds(vtsds);
  action       packed(1);
  title        char(48);
  cmd          char(512);
end-pi;

cmd      = vts.pcoCmd;
cmdLow   = lowercase(cmd);

if (%scan('.exe':cmdLow) > 0
or  %scan('.dll':cmdLow) > 0 )
and %scan('start': cmdLow) = 0;
  URL = fetchURL(cmd:':');

```

```

        if URL = '';
            URL = fetchURL(cmd:'\\');
        endif;

        cmd = URL;
    endif;

    return;

* ----- */

dcl-proc fetchURL;

dcl-pi *n varchar(512);
        cmd          char(512);
        searchValue   char(10) value;
end-pi;

dcl-s URL          char(512);
dcl-s URLStart     int(5);
dcl-s URLEnd       int(5);
dcl-s pos          int(5);

    URLStart = 0;
    URLEnd   = 0;
    pos = %scan(%trim(searchValue): cmd);

    if pos > 0;

        for c = pos-1 downto 1;
            if %subst(cmd:c:1) = '';
                leave;
            endif;
        endfor;

    endif;

    if c > 0;
        URLStart = c + 1;
        URLEnd   = 512 - URLStart;
        URL = %subst(cmd: URLStart: URLEnd);
    endif;

    return URL;

end-proc;

```

Virtual Terminal API (VT)

The RPG source code `iceCapTerm.rpg1e` provides detailed technical information about the syntax of the Virtual Terminal (VT) API. This code is not intended to be modified or recompiled.

```

**free
// -----
// Project . . . : IceBreak
// Design . . . : Niels Liisberg
// Function . . : 5250 Virtual Terminal wrapper for Ajax / JSON interface
//
// Copyright [2009] [System & Method A/S]
//
// By      Date      Task      Description
// -----
// NLI      02.01.2008 0000671 New program - Subproject IceCap
// -----
/if not defined(ICECAPTERM_DEF)
/define ICECAPTERM_DEF
// Constants
Dcl-C vtF1      const(x'31');
Dcl-C vtF2      const(x'32');
Dcl-C vtF3      const(x'33');
Dcl-C vtF4      const(x'34');
Dcl-C vtF5      const(x'35');
Dcl-C vtF6      const(x'36');
Dcl-C vtF7      const(x'37');
Dcl-C vtF8      const(x'38');
Dcl-C vtF9      const(x'39');
Dcl-C vtF10     const(x'3A');
Dcl-C vtF11     const(x'3B');
Dcl-C vtF12     const(x'3C');
Dcl-C vtF13     const(x'B1');
Dcl-C vtF14     const(x'B2');
Dcl-C vtF15     const(x'B3');
Dcl-C vtF16     const(x'B4');
Dcl-C vtF17     const(x'B5');
Dcl-C vtF18     const(x'B6');
Dcl-C vtF19     const(x'B7');
Dcl-C vtF20     const(x'B8');
Dcl-C vtF21     const(x'B9');
Dcl-C vtF22     const(x'BA');
Dcl-C vtF23     const(x'BB');
Dcl-C vtF24     const(x'BC');
Dcl-C vtSLP     const(x'3F');
Dcl-C vtFET     const(x'50');
Dcl-C vtPA1     const(x'6C');
Dcl-C vtPA2     const(x'6E');

```

```

Dcl-C vtPA3      const(x'6B');
Dcl-C vtClear    const(x'BD');
Dcl-C vtEnter    const(x'F1');
Dcl-C vtHelp     const(x'F3');
Dcl-C vtRollldown const(x'F4');
Dcl-C vtRollup   const(x'F5');
Dcl-C vtPrint    const(x'F6');
Dcl-C vtRecBS    const(x'F8');

// -----
// Types:
// -----

Dcl-DS vtsDS based(prototype_only) qualified;
  scnBuf      Char(3564); //total screen buffer
  scn80x24    Char(80)   dim(24) overlay(scnBuf);
  scn132x27   Char(132)  dim(27) overlay(scnBuf);
  width       Int(5); //width of the current screen typical 80 or 132
  height      Int(5); //height of the current screen typical 24 or 27
  cursorLin   Int(5); //current cursor location line number 1=First
  cursorCol   Int(5); //current cursor location coloumn number 1=First
  inputFields Int(5); //Number of input capable fields on the current screen
  focus       Int(5); //input field that has focus 1=First
  errLin      Int(5); //Line number where the error or errorsuble is
  winLevel    Int(5); //Each save/restore display can be used as system modal windows
  winWidth    Int(5); //Width of the current window in number of chars; 0=No window
  detected
  winHeight   Int(5); //Height of the current window in number of chars; 0=No window
  detected
  winCol      Int(5); //Starting coloun of data for the current window ; 0=No window
  detected
  winLin      Int(5); //Starting line number of data in the current window;0=No window
  detected
  winFound    ind;     // Window in 5250 stream
  winWdsf     ind;     // Window is defined by WDSF ( off=Attributes )
  winPulldown ind;     // Window is a pull down related to a mmenuBar
  winCntrl    Char(51); //Internal window control structure

  /*ON if you need to press enter after PCO reque
  ignoreErrMsgLine Ind; //after running the following command
  /*ON if you need to press enter after PCO reque
  pcoPressEnter Ind; //after running the following command
  pcoCmd        Varchar(200); //The PC organizer command if encountet
  messages      Ind; /*ON if messages is pending i display msg queue
  status        Int(10); //0=Ok; -1=Timeout >0 Automatic Sign-on Reason codes
  deviceName    Char(10); //Name of the conected device codes
  errMsg        Varchar(128); //If an error messages, this has the text else '
  closeAction   Ind;
  showScreen    Ind;

```

```

closeSession    Ind;
showMsgBox      Ind;
timeout         Int(5);
pollTimeOut     Int(5); // Timer to return data even if data is not complete ( status
will be -2 in that case
inviteTimeOut   Int(5); // When displayfiles contains ASSUME keyword, IceCap makes a
Lookahead. 0=No Lookahead, >= 1 seconds to wait
hasInvite       Int(5); // 0=No assume, 1=No input fields, 2=Any record formats
detectWindows   Int(5); // 0=never, 1=always , 2=native
colMin          Int(5); //Changes in from-coloumn since last complete
colMax          Int(5); //Changes in to-coloumn line since last complete
linMin          Int(5); //Changes in from-line since last complete I/O
linMax          Int(5); //Changes in to-line since last complete I/O

// -- Emulator scratch pad Not over 2048 Bytes and keep 512 bytes Leading
filler          Char(512);
// -- Scratch pad
sesNo           Int(10);
specChar        Char(4);
curlbeg         Char(1)   overlay(specChar:1);
curlend         Char(1)   overlay(specChar:2);
brabeg          Char(1)   overlay(specChar:3);
braend          Char(1)   overlay(specChar:4);
ilobUsed        Ind;
title           Varchar(127); //If an error messages, this has the text else '
i18nLearning    Ind; // *ON, set i18n Language breakout in capture mode
pParms          Pointer; // *ON, set i18n Language breakout in capture mode
userData        Char(1022); //User scratchpad: Use it as you like
End-DS;

```

```

Dcl-DS vtFieldDS based(prototype_only) qualified;
value           Varchar(3564);
col             Int(5);
lin            Int(5);
maxLen         Int(5);
dspAttr        Char(1);
ffw            uns(5);
fcw            uns(5);
metaData       Varchar(32767);
alias          Varchar(32);
rows           Int(5);
size           Int(5);
editmask       Varchar(32);
properties     Varchar(4096);
End-DS;

```

```

Dcl-DS vtLogonDS based(prototype_only) qualified;
userId         Char(10); //i5/OS User profile id or blanks
password       Char(32); //i5/OS password or blanks if sign on screen
programName    Char(10); //i5/OS Initial program to call or blanks

```

```

menuName      Char(10); //i5/OS Initial menu to show or blanks
currentLib    Char(10); //i5/OS current library or blanks if userprofil
deviceName    Char(10); //i5/OS virtual device name or blanks for auto
prfToken      Char(32); //i5/OS security profile handle or blanks
trace         Ind; //ON if 5250 trace is to be produced
tracePath     Varchar(128); //Filename and path for the trace file
keepInput     Ind; //ON if input fields is kept (Classic Scrape)
autoLogon     Ind; //ON - Logon for current user (userid must be set
ccsid         Int(5); //CCSID for emulator. 0=CCSID from current job
sesRel        Int(5); //get it with: num(getServerVar('SESSION_REL'));
timeOut       Int(5); //Number of seconds to wait for an I/O to
statusCallback Pointer(*PROC); //Pointer to callback procedure to recive status
filler        Char(512); //More to come..

```

End-DS;

```
Dcl-DS vtLocationDS based(prototype_only) qualified;
```

```
Lin      Int(5); //Line number (1=First)
```

```
Col      Int(5); //Coloumn number (1=First)
```

End-DS;

// Record format entry on display - IBM: Part of OINF0200 Format

```
Dcl-DS vtFormatEntryDS based(prototype_only) qualified;
```

```
dspRcfFmt      char(10); // Active record format name
```

```
dspRcfType     char(10); // Record format type
```

```
startRow       Int(10); // Start row
```

```
endRow         Int(10); // End row
```

```
windowStartColumn Int(10); // Window start column
```

```
windowEndColumn Int(10); // Window end column
```

End-DS;

// ALL record format on display - IBM: OINF0200 Format

```
Dcl-DS vtOutFormatsDS based(prototype_only) qualified;
```

```
bytesReturned  Int(10); // Bytes returned
```

```
bytesAvailable Int(10); // Bytes available
```

```
listOffset     Int(10); // Offset to the list of active record formats
```

```
activeFormats  Int(10); // Number of active record formats
```

```
entrySize      Int(10); // Size of an active record format entry
```

```
dspFileName    char(10); // Display file name
```

```
dspFileLib     char(10); // Display file library name
```

```
entries        likeds(vtFormatEntryDS) dim(64); // Array of enteries
```

End-DS;

// -----

// Proto types:

// -----

// Returns a pointer to a new Virtual Terminal Session (Keep that pointer

```
Dcl-PR vtOpen Pointer extproc(*CWIDEN:'vtOpen');
```

```
vtLogon        likeds(vtLogonDS) options(*NOPASS);
```

End-PR;

// When done using the session - always remember to close the session, whic

Dcl-PR vtClose Pointer extproc(*CWIDEN: 'vtClose');

pVts Pointer value; *//Pointer to Virtual Terminal Session*

End-PR;

// To ensure that client program sets data in the emulator with the same co

// net needed if client codepage is the same as emulator session codepage

// Make a roundtrip to the session with input fields, your key pressed, and

Dcl-PR vtPressKey extproc(*CWIDEN: 'vtPressKey');

pVts Pointer value; *//Pointer to Virtual Terminal Session*

Key Char(1) value; *//Key to press (See Consts)*

End-PR;

// Make a roundtrip to the session with input fields, your key pressed, and

// Here the key is converted from a text string into the key code

Dcl-PR vtPressFormKey extproc(*CWIDEN: 'vtPressFormKey');

pVts Pointer value; *//Pointer to Virtual Terminal Session*

keyName Pointer value options(*string); *//Name of the key like 'ENTER', 'F3'*

etc.

End-PR;

// Convert a Lin and Coloum to Location structure

Dcl-PR vtLinCol likeds(vtLocationDS) extproc(*CWIDEN: 'vtLinCol');

Lin Int(5) value; *//Line number of field*

Col Int(5) value; *//Coloumn number of field*

End-PR;

// Locates an input field and updates its value

Dcl-PR vtSetFieldNo extproc(*CWIDEN: 'vtSetFieldNo');

pVts Pointer value; *//Pointer to Virtual Terminal Session*

FieldNo Int(10) value; *//Index to field. 1=First*

Value Pointer value options(*string); *//New contents of field*

End-PR;

// Locates an input field and updates its value

Dcl-PR vtSetFieldNo2 extproc(*CWIDEN: 'vtSetFieldNo2');

pVts Pointer value; *//Pointer to Virtual Terminal Session*

FieldNo Int(10) value; *//Index to field. 1=First*

Value Pointer value options(*string); *//New contents of field*

len Int(10) value; *//Value Length*

End-PR;

// Locates an input from lin/col and update if it exists

Dcl-PR vtSetField extproc(*CWIDEN: 'vtSetField');

pVts Pointer value; *//Pointer to Virtual Terminal Session*

Lin Int(5) value; *//Line number of input field*

Col Int(5) value; *//Coloumn number of input field*

Value Pointer value options(*string); *//New contents of field*

End-PR;

// Locates an input from lin/col and update if it exists

```
Dcl-PR vtSetFieldLoc  extproc(*CWIDEN: 'vtSetFieldLoc');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    Location       likeds(vtLocationDS) value; //Lin/Col location
    Value          Pointer    value options(*string); //New contents of field
End-PR;
```

// Locates an field from lin/col datastructure and update if it exists

// The metadata is typically a JSON object that the emulator can intercept

```
Dcl-PR vtSetMetaData  extproc(*CWIDEN: 'vtSetMetaDataLoc');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    Location       likeds(vtLocationDS) value; //Lin/Col location
    metaData       Pointer    value options(*string); //additional userdefined options (i.e.
json)
End-PR;
```

// Locates an input field and returns is current value

// The field value, attributes, type, formatting etc is returned in the vtF

```
Dcl-PR vtGetField  extproc(*CWIDEN: 'vtGetFieldLoc');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    Location       likeds(vtLocationDS) value; //Lin/Col location
    vtFieldDS      likeds(vtFieldDS); //Structure to contain result
End-PR;
```

// The field value, attributes, type, formatting etc is returned in the vtF

```
Dcl-PR vtGetFieldNo  extproc(*CWIDEN: 'vtGetFieldNo');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    FieldNo        Int(10)    value; //Index to field. 1=First
    vtFieldDS      likeds(vtFieldDS); //Structure to contain result
End-PR;
```

// Locates an input field in a window and returns is current value

// The field value, attributes, type, formatting etc is returned in the vtF

```
Dcl-PR vtWinGetFieldNo  extproc(*CWIDEN: 'vtWinGetFieldNo');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    FieldNo        Int(10)    value; //Index to field. 1=First
    vtFieldDS      likeds(vtFieldDS); //Structure to contain result
End-PR;
```

// Set the cursor to a direct Line/col

```
Dcl-PR vtSetCursor  extproc(*CWIDEN: 'vtSetCursor');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    Lin            Int(5)     value; //Line number 1=First
    Col            Int(5)     value; //Coloumn number 1=First
End-PR;
```

// Set the cursor to a direct Line/col via Locations datastructure

```

Dcl-PR vtSetCursorLoc  extproc(*CWIDEN: 'vtSetCursorLoc');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    Location      likeds(vtLocationDS) value; //Lin/Col Location
End-PR;

// Set the cursor relative to current window or fullscreen panel
Dcl-PR vtWinSetCursor  extproc(*CWIDEN: 'vtWinSetCursor');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    Line          Int(5)     value; //Line number 1=First
    Col           Int(5)     value; //Coloumn number 1=First
End-PR;

// Set the cursor relative to current window or fullscreen panel
Dcl-PR vtWinSetCursorLoc  extproc(*CWIDEN: 'vtWinSetCursorLoc');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    Location      likeds(vtLocationDS) value; //Lin/Col Location
End-PR;

// Set the cursor to a position of a field number
Dcl-PR vtSetCursorFieldNo  extproc(*CWIDEN: 'vtSetCursorFieldNo');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    FieldNo       Int(10)    value; //Index to field. 1=First
End-PR;

// Find first occurent of an input/output field relative from the search st
// If the search argument was not found i returns *LOVAL (both line and col
//   FieldType must be one of the following:
Dcl-C VT_INPUT    const(0);
Dcl-C VT_OUTPUT   const(1);
Dcl-C VT_ANY      const(2);

//   Direction must be one of the following:
Dcl-C VT_NORTH    const(0);
Dcl-C VT_EAST     const(1);
Dcl-C VT_SOUTH    const(2);
Dcl-C VT_WEST     const(3);
Dcl-C VT_AT       const(4);

Dcl-PR vtLocate  likeds(vtLocationDS) extproc(*CWIDEN: 'vtLocate');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    fieldType     Int(5)     value;
    StartLocation likeds(vtLocationDS) value; //Lin/Col Location to start from
    direction     Int(5)     value;
    Search        Pointer    value options(*string); //String to searhc for
End-PR;

// Returns the current panel as a JSON object
// !! Depricated, use individual:
//   vtGetJsonScreenInfo, vtGetJsonInputFields, vtGetJsonOutputFields
Dcl-PR vtGetJSON  extproc(*CWIDEN: 'vtGetJSON');

```

```

    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    jsonStr       Varchar(32768) options(*varsize) ;
End-PR;

// Returns the current panel base info as JSON object
Dcl-PR vtGetJsonScreenInfo extproc(*CWIDEN: 'vtGetJsonScreenInfo');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    jsonStr       Varchar(32768) options(*varsize) ;
End-PR;

// Returns the current panel array of input fields as JSON object
Dcl-PR vtGetJsonInputFields extproc(*CWIDEN: 'vtGetJsonInputFields');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    jsonStr       Varchar(32768) options(*varsize) ;
End-PR;

// Returns the current panel array of input fields as JSON object
Dcl-PR vtGetJsonOutputFields extproc(*CWIDEN: 'vtGetJsonOutputFields');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    jsonStr       Varchar(32768) options(*varsize) ;
End-PR;

// Returns a substring of the current panel from a
// specified lin and col with respect to the current size 80x24 or 132x27
Dcl-PR vtGetScreen Varchar(4096) extproc(*CWIDEN: 'vtGetScreen');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    Line          Int(10)    value; //Line number on screen to retrieve (1=First)
    Coloumn       Int(10)    value; //Coloumn number on screen to retrieve (1=First)
    Length        Int(10)    value; //Length of data to return - Line wraps can occur if
end is > line
End-PR;

// Returns a substring of the current panel from a
// specified lin and col with respect to the current size and location of a
// If no window is active it will returnsame value as vtGetScreen
Dcl-PR vtWinGetScreen Varchar(4096) extproc(*CWIDEN: 'vtWinGetScreen');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    Line          Int(10)    value; //Line number on screen to retrieve (1=First)
    Coloumn       Int(10)    value; //Coloumn number on screen to retrieve (1=First)
    Length        Int(10)    value ; //Length of data to return - Line wraps can occur if
end is > line
End-PR;

// Returns true / *ON if the value argument was found
// in the pannel Title whichs in turn must comply to the SAA standart:
// 1) The title is assumed to be centred.
// 2) It must be surround by highlight / white attributes
Dcl-PR vtSaaTitleContains Ind extproc(*CWIDEN: 'vtSaaTitleContains');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session

```

```

    Value          Pointer    value options(*string) ; //value to search for
End-PR;
// Returns the title of an SAA compilent panel
// 1) The title is assumed to be centred in the first line
// 2) It must be surround by highlight / white attributes
Dcl-PR vtSaaTitle Varchar(256) extproc(*CWIDEN: 'vtSaaTitle');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
End-PR;

// Returns true / *ON if the value argument was found
// in the Window Title which in turn must comply to the SAA standart:
// 1) The title is assumed to be centred.
// 2) It must be surround by highlight / white attributes
Dcl-PR vtWinTitleContains Ind extproc(*CWIDEN: 'vtWinTitleContains');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    Value          Pointer    value options(*string) ; //value to search for
End-PR;

// Returns the title of an SAA compilent Window - if at window is active
// Otherwise the titel of the panel
// 1) The title is assumed to be centred at the first line in the window
// 2) It must be surround by highlight / white attributes
Dcl-PR vtWinTitle Varchar(256) extproc(*CWIDEN: 'vtWinTitle');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
End-PR;

// Returns a given line number from the current window.
// If no window is active, the line from the current display is returned
// The line is stripped for display attributes
Dcl-PR vtWinGetLine Varchar(256) extproc(*CWIDEN: 'vtWinGetLine');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    LineNo        Int(5)     value;
End-PR;

// Returns a given line number from the current window.
// If no window is active, the line from the current display is returned
// The line is returned "as is" with display attributes and non prinable ch
Dcl-PR vtWinGetLineRaw Varchar(256) extproc(*CWIDEN: 'vtWinGetLineRaw');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    LineNo        Int(5)     value;
End-PR;

// Patch a give string into the screen buffer in the relative window
// In the final buffer after decorartion
// can also be used to remove data, by inserting blanks
Dcl-PR vtWinPatch extproc(*CWIDEN: 'vtWinPatch');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    Line          Int(5)     value; //Line number in relative window 1=First line
    Coloumn       Int(5)     value; //Coloumn number in relative window 1=First
    value         Varchar(256) options(*VARSIZE) const;

```

```

padding      Int(5)      value options(*nopass); //Total Length to patch
End-PR;

// Patch a give string into the screen buffer in the relative window
// In the raw 5250 stream before any decoration
// can also be used to remove data, by inserting blanks
Dcl-PR vtWinPatchRaw extproc(*CWIDEN: 'vtWinPatchRaw');
  pVts        Pointer    value; //Pointer to Virtual Terminal Session
  Line        Int(5)     value; //Line number in relative window 1=First line
  Coloumn     Int(5)     value; //Coloumn number in relative window 1=First
  value       Varchar(256) options(*VARSIZE) const;
  padding     Int(5)     value options(*nopass); //Total Length to patch
End-PR;

// Return the List of output data for the corrent window / of screen
Dcl-C VT_FIRST const(0);
Dcl-C VT_NEXT  const(1);
Dcl-PR vtGetOutputList Ind extproc(*CWIDEN: 'vtGetOutputList');
  pVts        Pointer    value; // Pointer to Virtual Terminal Session
  position     Int(5)     value; // The position in the list: VT_FIRST, VT_NEXT
  line        Int(5);     // Line number in the window
  col         Int(5);     // Coloumn number in the window
  offset      Int(5);     // offset for start of field where data begins
  length      Int(5);     // total lengt to display including blanks
  attr        Char(1);    // display attribute > 0x20 and < 0x40
  data        Varchar(3564) options(*VARSIZE); //the output text
End-PR;

// Return the List of formats on screen
Dcl-PR vtGetOutputFormatList extproc(*CWIDEN: 'vtGetOutputFormatList');
  pVts        Pointer    value; // Pointer to Virtual Terminal Session
  OutFormats   liked(vtOutFormatsDS); // Formats on display
End-PR;

// Reload the output buffer (ScnBuf in the vts structutre) i.e. for vtGetOutputList()
Dcl-PR vtReloadOutputBuffer extproc(*CWIDEN: 'vtReloadOutputBuffer');
  pVts        Pointer    value; //Pointer to Virtual Terminal Session
End-PR;

// Handy wrapper - get the input value( the east value) for at Label
Dcl-PR vtGetValueFor Varchar(1024) extproc(*CWIDEN: 'vtGetValueFor');
  pVts        Pointer    value; //Pointer to Virtual Terminal Session
  label       Pointer    value options(*string); //value to search for
End-PR;

Dcl-PR vtSetCcsid Pointer extproc(*CWIDEN: 'vtSetCcsid');
  pVts        Pointer    value; //Pointer to Virtual Terminal Session
  ccsid       Uns(5)     value; //1-65535

```

End-PR;

```
Dcl-C VT_CLIENT2VT const(0);
//xlata descriptor from virtual terminal to clie
Dcl-C VT_VT2CLIENT const(1);
//xlata descriptor from client to virtual termin
```

```
Dcl-PR vtGetXlateDesc Pointer extproc(*CWIDEN: 'vtGetXlateDesc');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    type          Uns(5)     value; //
End-PR;
```

```
// Util: Convert to readable hex
Dcl-PR vtHex Varchar(10) extproc(*CWIDEN: 'vtHex');
    Value          Int(10)    value;
End-PR;
```

```
// Util: Convert to readable hex
Dcl-PR vtHexChar Varchar(10) extproc(*CWIDEN: 'vtHexChar');
    Value          Char(1)    value;
End-PR;
```

```
// Util: trim leading unprintable , return offset to first non blank
Dcl-PR vtTrim Int(10) extproc(*CWIDEN: 'vtTrim');
    Outparm        Varchar(256) options(*varsize);
    Inparm         Varchar(256) options(*varsize);
End-PR;
```

```
// Text to place in the trace
Dcl-PR vtPrintf extproc(*CWIDEN: 'vtPrintf');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    formatStrinf   Pointer    value options(*string); //format to printf
    t1            Pointer    value options(*string:*nopass); //string constatnt
    t2            Pointer    value options(*string:*nopass); //string constatnt
    t3            Pointer    value options(*string:*nopass); //string constatnt
    t4            Pointer    value options(*string:*nopass); //string constatnt
    t5            Pointer    value options(*string:*nopass); //string constatnt
End-PR;
```

```
// Text to place in the joblog
Dcl-PR vtJoblog extproc(*CWIDEN: 'vtJoblog');
    formatStrinf   Pointer    value options(*string); //format to printf
    t1            Pointer    value options(*string:*nopass); //string constatnt
    t2            Pointer    value options(*string:*nopass); //string constatnt
    t3            Pointer    value options(*string:*nopass); //string constatnt
    t4            Pointer    value options(*string:*nopass); //string constatnt
    t5            Pointer    value options(*string:*nopass); //string constatnt
End-PR;
```

```

// post mortem, debug via a trace file; only supply the 5250 stream
// three lines
Dcl-PR vtParseTrace  extproc(*CWIDEN: 'vtParseTrace');
    pVts          Pointer    value; //Pointer to Virtual Terminal Session
    traceFile      Pointer    value options(*string); //path and filename to partial trace
file
End-PR;
// Util: returns the device status:

// Value Definition
// -1      Does not exists
// 0       VARIED OFF - The system is not using the description.
// 10      VARY OFF PENDING - The description is being varied off. During t
// 20      VARY ON PENDING - The description is being varied on. During thi
// 30      VARIED ON - The tasks that manage the network interface, network
// 32      VARY ON/CNN PENDING - The first of a pair of OptiConnect control
// 40      CONNECT PENDING - This status is only valid for switched SDLC, I
// 50      SIGNON DISPLAY - This status is only valid for display devices.
// 51      ACTIVE/CNN PENDING - The first of a pair of OptiConnect controll
// 60      ACTIVE - The object is successfully placed in VARIED ON status.
// 63      ACTIVE READER - The device is in use by a spool reader.
// 66      ACTIVE WRITER - The device is in use by a spool writer.
// 67      AVAILABLE - The independent auxiliary storage pool (ASP) device
// 70      HELD - This status is only valid for Device Descriptions. The us
// 80      RCPYND - Error recovery is pending for the line, controller, or
// 90      RCYCNL - Error recovery is canceled for the network interface, l
// 95      SYSTEM REQUEST - The display device has been requested by the sy
// 100     FAILED - An error occurred for the network interface, network se
// 103     FAILED READER - An error occurred for the device while in use by
// 106     FAILED WRITER - An error occurred for the device while in use by
// 107     SHUTDOWN - The NWSD was shut down using an Application Program I
// 110     DIAGNOSTIC MODE - The network interface, network server, line, c
// 111     DAMAGED - The network interface, network server, line, controle
// 112     LOCKED - The actual status of the resource cannot be determined
// 113     UNKNOWN - The status indicator of the description cannot be dete

Dcl-C VT_NOT_EXSISTS const(-1);
Dcl-C VT_VARIED_OFF const(0);
Dcl-C VT_VARY_OFF_PENDING const(10);
Dcl-C VT_VARY_ON_PENDING const(20);
Dcl-C VT_VARIED_ON const(30);
Dcl-C VT_VARY_ON_CNN_PENDING const(32);
Dcl-C VT_CONNECT_PENDING const(40);
Dcl-C VT_SIGNON_DISPLAY const(50);
Dcl-C VT_ACTIVE_CNN_PENDING const(51);
Dcl-C VT_ACTIVE const(60);
Dcl-C VT_ACTIVE_READER const(63);
Dcl-C VT_ACTIVE_WRITER const(66);
Dcl-C VT_AVAILABLE const(67);
Dcl-C VT_HELD const(70);

```

```

Dc1-C VT_RCYPND    const(80);
Dc1-C VT_RCYCNL    const(90);
Dc1-C VT_SYSTEM_REQUEST const(95);
Dc1-C VT_FAILED    const(100);
Dc1-C VT_FAILED_READER const(103);
Dc1-C VT_FAILED_WRITER const(106);
Dc1-C VT_SHUTDOWN  const(107);
Dc1-C VT_DIAGNOSTIC_MODE const(110);
Dc1-C VT_DAMAGED   const(111);
Dc1-C VT_LOCKED    const(112);
Dc1-C VT_UNKNOWN   const(113);

Dc1-PR vtDeviceStatus Packed(5:0) extproc(*CWIDEN: 'vtDeviceStatus');
      deviceName      Char(10)  const; //Name of display device
End-PR;

Dc1-PR vtIsIceCap ind extproc(*CWIDEN: 'vtIsIceCap');
End-PR;

// ----- field level API -----
// DDS data type
// Signed numeric - DDS type 'S'
// Numeric shift - DDS type 'N'
// Numeric only - DDS type 'Y'
// Digits only - DDS type 'D'

Dc1-PR vtFfwDDSDataType char(1) extproc(*CWIDEN: 'vtFfwDDSDataType');
      vtField          likeds(vtFieldDS); // Structure containing field
End-PR;

Dc1-PR vtFfwIsProtected ind extproc(*CWIDEN: 'vtFfwIsProtected');
      vtField          likeds(vtFieldDS); // Structure containing field
End-PR;

Dc1-PR vtFfwIsDupkey ind extproc(*CWIDEN: 'vtFfwIsDupkey');
      vtField          likeds(vtFieldDS); // Structure containing field
End-PR;

Dc1-PR vtFfwIsModifiedDataTag ind extproc(*CWIDEN: 'vtFfwIsModifiedDataTag');
      vtField          likeds(vtFieldDS); // Structure containing field
End-PR;

// Field Shift/Edit Specification - unmaped
Dc1-C VT_FFW_ALPHA_SHIFT      const(0);
Dc1-C VT_FFW_ALPHA_ONLY      const(1);
Dc1-C VT_FFW_NUMERIC_SHIFT    const(2);
Dc1-C VT_FFW_NUMERIC_ONLY     const(3);
Dc1-C VT_FFW_KATAKANA_SHIFT    const(4);
Dc1-C VT_FFW_DIGITS_ONLY      const(5);
Dc1-C VT_FFW_INPUT_INHIBIT    const(6);

```

```
Dcl-C VT_FFW_SIGNED_NUMERIC      const(7);

Dcl-PR vtFfwDataType int(5) extproc(*CWIDEN: 'vtFfwDataType');
    vtField          likes(vtFieldDS); // Structure containing field
End-PR;

Dcl-PR vtFfwIsAutoEnter ind extproc(*CWIDEN: 'vtFfwIsAutoEnter');
    vtField          likes(vtFieldDS); // Structure containing field
End-PR;

Dcl-PR vtFfwIsFieldExitRequired ind extproc(*CWIDEN: 'vtFfwIsFieldExitRequired');
    vtField          likes(vtFieldDS); // Structure containing field
End-PR;

Dcl-PR vtFfwIsUppercase ind extproc(*CWIDEN: 'vtFfwIsUppercase');
    vtField          likes(vtFieldDS); // Structure containing field
End-PR;

Dcl-PR vtFfwIsReserved ind extproc(*CWIDEN: 'vtFfwIsReserved');
    vtField          likes(vtFieldDS); // Structure containing field
End-PR;

Dcl-PR vtFfwIsMandatoryEnter ind extproc(*CWIDEN: 'vtFfwIsMandatoryEnter');
    vtField          likes(vtFieldDS); // Structure containing field
End-PR;

Dcl-PR vtFfwIsNoAjust ind extproc(*CWIDEN: 'vtFfwIsNoAjust');
    vtField          likes(vtFieldDS); // Structure containing field
End-PR;

Dcl-PR vtFfwIsRightAjustZeroFill ind extproc(*CWIDEN: 'vtFfwIsRightAjustZeroFill');
    vtField          likes(vtFieldDS); // Structure containing field
End-PR;

Dcl-PR vtFfwIsRightAjustBlankFill ind extproc(*CWIDEN: 'vtFfwIsRightAjustBlankFill');
    vtField          likes(vtFieldDS); // Structure containing field
End-PR;

Dcl-PR vtFfwIsMandatoryFill ind extproc(*CWIDEN: 'vtFfwIsMandatoryFill');
    vtField          likes(vtFieldDS); // Structure containing field
End-PR;

/endif
```