USER GUIDE

Presto Release 10.0.
For IBM i and Windows.

Presto™

Instantly Web Enable and Enhance IBM i 5250 Applications and Screens

Presto extends the life of your 5250 applications
by helping you rapidly modernize your IBM i.

Fresche Solutions
995 Wellington, Suite #200
Montreal, Quebec
Canada, H3C 1V3

Phone: (514) 747.7007
(toll-free in US and Canada): 1.800.361.6782
(toll-free in Belgium, France, Germany, UK):
00 800 361 67 82 0
(toll-free in Australia): 0011 800 361 6782 0

E-Mail: info@freschesolutions.com
Web: www.freschesolutions.com/presto

Technical Support:
101-9724 Fourth Street
Sidney, BC Canada V8L 2Y7
Phone: 250-655-1766
Fax: 250-655-1733
E-Mail: support@freschesolutions.com
Web: www.myFreschesolutions.com

Copyright Notice

Presto 10.0 program © 2024 Fresche Solutions Inc.

Presto 10.0 User Guide. Copyright © 2024. All rights reserved. This document may not, in whole or part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without written consent from Fresche Solutions Inc.

System Requirements

Presto requires a minimum OS/400 release level of V7R3. You also need to have TCP/IP and the IBM HTTP Server (Powered by Apache) configured correctly. In order to run fully in batch you must install the WebSphere V6R0 Application Server. Note that only the WebFacing server needs to be running.

Presto will initially occupy approximately 20 MB on your IBM i, and up to 70 MB at times during the install and until the install libraries are removed from your system.

The Presto Designer will run on any PC running at least Windows 8 and .NET Framework 4. Your Presto users will need Windows 8 or higher, with a modern browser (we test with Chrome, Firefox, IE9+). Note that while we test Presto on specific versions, the major browsers by default auto-update themselves, which means that we cannot guarantee that the version of these browsers that you use to run Presto are the same ones we test with. We do expect things to run well, but periodically quirks do surface.

Acknowledgments

Throughout this manual, reference is made to several trademarks: Presto is trademark of Fresche Solutions Inc. IBM, AS/400, iSeries, IBM i and OS/400 are trademarks of International Business Machines. Microsoft®, Windows® and Word® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Adobe® and Flash® are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. All other trademarks are acknowledged as the properties of their respective owners.

Creation date (PDF) February 16, 2024

About this Guide

This is the Presto 10.0 User Guide. Here you will find instructions for getting started with Presto as quickly as possible as well as reference information for developers using Presto on a daily basis.

Below is an overview of the guide and recommendations on how to use it.

Installation Instructions

To install Presto for the first time follow the instructions in "Installation and Configuration" on page 7 discusses the system requirements and objects that are installed and guides you through the steps to get the software up and running.

If you are upgrading to Presto 10.0 from an earlier version please follow the installation instructions in the Upgrader's Guide.

The Presto Tutorial

The tutorial application includes a menu, several subfile programs, two windows and a number of data panels for working with individual records.



You can now find the tutorial for Presto at our website, at the following address:

```
http://www.myfreschesolutions.com/nexus/apps/PRTUTRL.pgm
```

We recommend you visit this page and follow the tutorial examples, labs, and videos.

The tutorial includes illustrated step-by-step instructions that will teach you how to work with the design tools available in Presto.

Presto Reference

This section includes reference information for all the features, tools and options available in Presto.

Appendices

The appendices address a number of more technical topics, such as an overview of the IBM i jobs that are related to Presto and an HTML tutorial written specifically for green screen programmers.

Find Tips & Techniques and Examples Online!

This guide will help you use the key features in the software, but we are constantly adding new examples, technical tips and videos to our online, searchable Knowledge Base and Code Samples.

You can find our online Knowledge Base at https://www.myfreschesolutions.com/kb, and the most recent Presto articles are here:

https://www.myfreschesolutions.com/kb?Product=Presto

Go to this address to access the Presto code examples and videos:

https://www.myfreschesolutions.com/nexus/ui/docs?docId=589

Getting Help

If you have any questions please call our support line at 250-655-1766 [7:30 - 5:00 PST] or start an online chat from any of the support pages at our site. You can also reach us by email at support@freschesolutions.com. We fully expect you to have the best support experience you've ever encountered.

You will also find it useful to familiarize yourself with the support links, online knowledge base and other resources at www.myfreschesolutions.com.

Live Product Demos

Our Sales and Support teams are happy to provide live product demos. If you want to get going as quickly as possible ask your sales rep to schedule a Presto demo for you and your development team.

Table of Contents

About this Guide ... 3
Table of Contents ... 5

Section 1 - New Features and Upgrade Guide

CHAPTER 1 - NEW FEATURES LIST AND UPGRADE CONSIDERATIONS ... 22

Presto 10.0 New Features ... 23
Upgrade Considerations ... 23
Why You Should Start an Upgrade by Cloning your Current Environment ... 24
Upgrade StrPcCmd Utility ... 24
Page Options Inheritance ... 24
Changes to Page-level custom Page Options ... 25
Changes to Element Height and Width ... 26
Dark Mode ... 26
JQuery Update ... 26
Transformation Position Adjusted Flag ... 26
Apache Changes for Presto REST API ... 26
Apache Changes for Presto Server API ... 27
License Key Change for Version 8 and Higher ... 27
License Key Enforces Seat Count ... 28
New License Server ... 28
Mitigate Some Security Risks by Not Allowing Access to the Uploads Folder ... 29
ParameterVersion Updated ... 30
Exodus55 ... 30

CHAPTER 2 - UPGRADING TO PRESTO 10.0 ... 32

Upgrade Overview ... 33
Step 1: Download Presto and Transfer the IBM i Install Files ... 35
Step 2: Install the Server Components ... 36
Step 3: Set your CCSID in the HTTP Server Configuration ... 40
Step 4: Start the HTTP Server Instance ... 40
Step 5: Confirm your Server Installation ... 42
Step 6: Install the Presto Designer ... 42
Step 7: Create a Connection to your New Environment ... 43
Step 8: Make any Required Post-Installation Changes, and Test ... 45

Step 9: Promote the Upgraded Environment(s) to Production 46
Creating New Development Environments 47
Using WebFacing with Presto 48
More About the Presto IBM Apache HTTP Server Instance 48
Troubleshooting 49

CHAPTER 3 - PRESTO NEW FEATURES HISTORY 51
Presto 9.9 New Features 51
Presto 9.8 New Features 52
Presto 9.7 New Features 52
Presto 9.6 New Features 53
Presto 9.5 New Features 54
Presto 9.4 New Features 54
Presto 9.3 New Features 55
Presto 9.2 New Features 55
Presto 9.1 New Features 56
Presto 9.0 New Features 56
Presto 8.9 New Features 57
Presto 8.8 New Features 58
Presto 8.7 New Features 59
Presto 8.6 New Features 59
Presto 8.5 New Features 60
Presto 8.4 New Features 62

Section 2 - Introduction and Installation

CHAPTER 4 - INTRODUCTION TO PRESTO 2
Benefits of Using Presto 2
How Presto Works 4
Key Features 5
Get More Out of Presto with other Fresche Solutions 6

CHAPTER 5 - INSTALLATION AND CONFIGURATION 7
Installation Overview 7
Starting the XL_PRESTO Apache HTTP Server Instance After an IPL 11
Starting the Exodus55 Subsystem After an IPL 12
Starting the License Server Subsystem After an IPL 13
Datepicker Localization 20

Moving On 23

Section 3 - Presto Reference

CHAPTER 6 - WORKING WITH THE PRESTO DESIGNER 33
Presto Out of the Box 33
Understanding the Range of Customization Features in Presto 33
Summary 35
Launching the Presto Designer 36
Configuring Your Environment(s) 37
Connection Settings 37
Server Settings 38
Detection Settings 40
Global Settings 42
IDE Settings 48
Skins 49
Advanced Configuration Options 53
Using the Presto Designer 63
Editing the Page Description 63
Testing Pages in a Browser 64
Using the Find Window 65
Changing the Layout of the Presto Designer 66
The Active Page Tab 67
The Screen View 67
Working with Page and Global Identities 68
Device Preview Dropdown 70
Working with Custom Files 71
Support for Saving Files Locally 71
Managing Pages and Customizations 73
The Pages Tab 73
Working with Customizations 74
Attaching Custom HTML, JavaScript or Transformations to a Page 75
Creating and Working with SQL Queries 77
Preview Query Results 79
General Considerations for Working with SQL Queries 80
Using SQL Queries in Custom HTML 80
Support for Dynamic Library and File Names in SQL 81

Using jQuery Selectors for SQL Parameters 82
Recording and Playing Macros 84
Recording a Macro in the Presto Designer 84
Replaying Macros in the Presto Designer 85
Playing a Macro from a Page 85
Managing and Editing Macros 86
Creating and Working with Auto Logins 88
Technical Requirements 88
Creating an Auto Login 89
How to Use Auto Logins 90
More About the Auto Login Template Shipped with Presto 92
Spool File Listings 93
Technical Details 95
Adding a Site Favicon 96
Creating a New File 97
Rotate Orientation of Display Mode 98
Access PC Applications from Your Presto Site 99
Requirements 99
How to Use this Feature 99
Dual Maintenance Management 105
How does it work? 105
Working with Display-File Changes 106
Page Up and Down Buttons 107
Submit, Cancel, Exit Buttons 108
Window and Table Text 109
Unique Naming of Customizations 110
Renaming a Customization 110
Naming Conflicts 111
Supported Characters 111
Customization Not Found 112
Export Customizations without the Contents 112
Considerations for Upgrading and Existing Customizations 113
Auto Detection Settings and Identities 114
Understanding Your Detection Settings 114
Why Use 'Auto' Detection Settings? 116
How Auto Panel Detection Settings Work 116

Window Detection 117
Potential Trouble Spots 117
Troubleshooting Page Identity Problems 119
Using Panel Identities to ID Screen Variations 122
Using Global Level Identities 124
Pages that Match More Than One Global Identity 132
Disable Page Identities 133
A Final Word of Caution 134

CHAPTER 7 - CUSTOMIZING WITH THE PAGE PROPERTIES 135

Working with Page Properties 136
Page Properties 137
Function Key Type & Text Options 137
Detect Menu Links 138
Adjust Last Row 139
Auto Formatting 139
Bypass Output Only Screen 140
Polling Interval (Seconds) 140
Detect Subfile Options 141
Hide Subfile Options 141
Table Detection and Custom Tables 143
Creating and Working with Custom Tables 144
Setting the Table Perimeter 146
Setting the Table Header 146
Adding Table Columns 146
Reviewing Your Changes 147
Working with Windows 147
Working with Window Options 148

CHAPTER 8 - GLOBAL RULES FOR PAGE OPTIONS 150

Global Rules for Page Options Manager 151
Working with Global Rules for Page Options 152
Anatomy of a Matching Criterion 154
How to create A Global Rule 155
Migrating Global Rules for Page Options 156

CHAPTER 9 - THE PRESTO VISUAL EDITOR 157

Using the Visual Editor 157

Selecting Page Elements 159
Moving Elements 160
Aligning Multiple Elements to a Common Baseline 161
Working with Element Properties 163
Reviewing, Resetting, Importing and Exporting Properties 163
Setting the Element Type 166
Transformation List Filtering 168
Resizing Elements 169
Example: Shortening Text Fields Using the Element Properties 169
Example: Using the Element Handle and Edges 170
Inserting New Elements 172
Selecting Multiple Items (Group Selection) 174
Editing Tables 175
Using Screen Values in the Visual Editor 176
Example 1: Using Screen Values as Autocomplete Parameters 176
Visual Editor Insert Elements Palette 179
Adding Custom Palette Entries 180
Reattach Transformations to Moved Elements 182
Copy Transformations between Pages 183

CHAPTER 10 - GENERAL TRANSFORMATION PROPERTIES 186

Changing Colors 187
Selecting a File 188
Events 189
Example: Triggering an Alert When a User Mouses Over an Element 189
Field Settings 191
HTML5 Input Types 193
The Class Attribute 194
Identification Properties 195
Padding Properties 196
Position Properties 197
Text Properties 198
Copy Attributes 199

CHAPTER 11 - VISUAL EDITOR TRANSFORMATIONS 200

Autocomplete 201
Switching a Field to an Autocomplete 201
Setting a Minimum Number of Characters to Enter Before Querying 202

Creating and Attaching a Query 203
Query Examples 205
jQuery Autocomplete Events 208
Block Container (div) 210
Adding Block Containers and Using the layer_N Class 210
Button 214
Function Key 215
Check Boxes 218
Data Export 219
Adding the Element to the Screen 219
Data Export Field Properties 219
Data Source Properties 220
Defining the Data Source SQL 221
Date Picker Field 223
Set Datepicker Options Based on Browser Language 225
Drop Down List 226
Data Source Options 226
Element Settings 226
Static Example: Returning the Displayed Value 227
Static Example: Displaying a Description for a Returned Code 228
Dynamic Example: Using SQL Queries to Generate Lists 229
Adding Static Values to Dynamic Lists 232
Blank Option for Dropdown 233
Google® Maps 235
Adding Markers 237
Linking a Map Snapshot to a Full Featured Google Map 238
Icon 239
iFrame 240
Image 241
Inline Container (span) 242
Label 243
Link 244
Password Field 245
Radio Buttons Transformation 246
Switching a Field to Radio Buttons 246
Creating a Static List of Options 247

Setting Dynamic Options 248
Signature Pad 250
SmartCharts 253
Chart Properties 253
Spinner 255
Spinner Properties 255
Tabs 257
Table 262
Text Area 263
Text Field 264

CHAPTER 12 - GLOBAL TRANSFORMATIONS 265

Global Transformation Manager 265
Adding a new Global Transformation from the Visual Editor 273
Migrating Global Transformations 273
Considerations 277
One Transform Event 278

CHAPTER 13 - ENHANCING YOUR PAGES WITH JAVASCRIPT 280

Adding Simple JavaScript to Presto Pages 281
Where To Create Your Customizations 281
Presto JavaScript Snippets 281
Hello World 283
Page HTML Visible via Dockable Window 284
Example: Adding a New Command Key 286
Adding a New Drop Down List of Commands 288
Adding an Event to a Drop Down Box 289
Adding Features to Your Datepickers 291
Using a Script to Add Dynamic Images to a Page 294
Adding SmartCharts with Custom JavaScript 296
Building a Dynamic Chart 296
Building a Chart Using Static Data 297
Building a Chart Using Screen Values 298
Support for Retrieving Dynamic Values from the Screen 299
Enhancing Autocompletes Manually 301
Using an External Web Program or Service to get Data for a Drop Down Box 303
Using the File custom.js 307
Datepicker Localization 307

Enabling an Alarm Sound 307
Turning Off the Message Waiting Indicator 307
Suggested Structure for Additional custom.js Enhancements 307
Add Key Mapping 308
Using a Transformation to Upload Files 309
Retrieve Job Info 310
SentRequest Event 311
PlayMacro Event 312

CHAPTER 14 - CREATING AND USING CUSTOM HTML 313

Creating New Custom HTML Entries 314
The Apprentice's Guide to Editing the HTML 316
Review the Basic HTML 316
Other Basic Panel Elements 318
The Presto Screen, Field and Attr [Attribute] Tags 320
Using Screen, Field and Attribute Tags 322
Generating Dynamic Screen Tags for All Text Constants 324
Considerations For Using Custom HTML 325
Optimizing the Initial HTML 325
Managing Static Text vs. Output-Only Fields in Custom HTML 326
Adding Green Screen Content to Your HTML 329

CHAPTER 15 - WORKING WITH YOUR PRESTO SKINS 331

About Presto Skins 331
Skin Support for Mobile Devices 332
Using jQuery Mobile 334

CHAPTER 16 - USING PRESTO WITH RPG OA 335

How it Works 335
Considerations 336
Important Performance Consideration for Upgraders 337
RPG Program Requirements 337
Customizing RPG OA Programs with Presto 339
About the jqGrid jQuery Grid Plugin 341
The Presto OA Grid API 342
Presto.Oa.Grid.GetField('FieldName') 342
Presto.Oa.Grid.SetSize(width, [height]) 342
Presto.Oa.Grid.SetSortable(bool) 342

Presto.SetScreenTitle('Title') 342
Example 342
Demo OA Program 343

CHAPTER 17 - THE PRESTO SKIN EDITOR 344
About the Skin Editor 344
Using the Skin Editor 344
Using Preset Themes 347
Making Changes to Preset Themes 348
Changing the Skin Logo 350
Color Options in the Skin Editor 353
Logo Options in the Skin Editor 353

CHAPTER 18 - WORKING WITH MOBILE CUSTOMIZATIONS 354
About the Mobile Customizations 354
How Mobile Mode Works 354
Enabling Mobile Customizations 355
Previewing Your Mobile-Specific Customizations 357
Page Customizations 359

CHAPTER 19 - USING THE RESPONSIVE VISUAL EDITOR 360
About the Responsive Editor 360
Understanding the Responsive User Interface 361
Activating Responsive Features 363
Change the Width of the Grid 363
Placing Elements in the Grid 364
Placing an Element: Drag & Drop Example 364
Position Properties 364
Resizing Elements 365
Saving Visual Editor Changes 366
Adding Responsive Grid 366
Adding a Cell 367
Adding a Row 368
Merging Cells 370
How to Select a Row 371
Removing a Cell 371
Removing a Row 372
Splitting a Cell 373

Visible On Property 373
Hidden On Property 374
Placing an Element: Highlighting 374
About Creating Responsive Grids 375
Considerations 375
Additional Features 375
Responsive CSS Classes 376
Example 376
Example 376

Section 4 - Managing Applications

CHAPTER 20 - PRESTO CHANGE MANAGEMENT 379
Customization Categories 380
Overview of the Migration Process 380
Managing Environment Level Enhancements 381
Skins 381
JavaScript Enhancements in custom.js 382
SQL Queries 383
Recorded Macros 383
Site Configuration Settings 383
Managing Page Level Customizations 384
Environment Configuration Settings 384
Exporting Customized Pages 384
Importing Pages to Another Environment 386

Section 5 - Appendices

APPENDIX A - ADDITIONAL PRESTO INSTALLATION OPTIONS 394
Configuring Multiple Installations 394
Installing a New Version 394
Creating a New HTTP Server Instance 395
Creating a New Server Connection 395
Renaming a Presto Library 396
Uninstalling a Presto Library 396
Using WebFacing with Presto 397
Virtual Terminals and CCSID Settings 398
Configuring Presto for Multi-Language Support 399

Working With the Presto Apache HTTP Server Instance 401

APPENDIX B - ADVANCED TECHNICAL TOPICS & TROUBLESHOOTING 403

Troubleshooting Presto Communications Issues 403
Rebuilding the PR_CONVQ Data Queue 404
How to Capture a Presto Debug Log 406
Designer Access to Server Logs 406
The Presto IBM i Jobs 407
Activating Polling Support 408
External Validation of Presto Session ID & User 409
Customizing Function Key Text Recognition 410
Subfile Option Flyout Customizations 411
Subfile Option Separator 411
Setting Detection Exceptions 411
Datepicker Auto-Detection 412
Presto Launch and URL Parameters 414
Custom Subsystem 416
Access to Page-Level JavaScript customizations in Firefox and Chrome Developer Tools 417
Multiple Data Queues 418
Bypassing Display Program Messages Screen 418
Load Testing with JMeter 419
How to create a Custom Error Handler 419

APPENDIX C - PRESTO.TRANSFORM FUNCTION REFERENCE 421

The Transform, Insert and Select Methods 421
Presto.Transform Support for Cascading Dropdown Lists 422
JavaScript Objects Used for the Transform and Insert Methods 423
Transformation Properties 425
to_object, object 425
prefix, suffix 429
source 430
value 431
Any Valid HTML Attribute 431
src 432
css 432
options 432
text_pos 433
Events Property 433

Presto Autocomplete Transformation Properties 435
Presto menulinks Transformation Properties 436
target 436
hide_value 436
hide_target 437
pattern_label, pattern_value, pattern_label_group and pattern_value_group 437
format 438
Customizing MenuLinks 438
Presto clickableoptions Transformation Properties 439
keep_target 439
Presto fileupload Transformation Properties 440
Presto Flyout Transformation Properties 441
JavaScript Functions to Retrieve and Set Presto Variables 442
APPENDIX D - PRESTO.EXECUTESQL 444
Technical Details 444
Parameters 444
Presto.OpenSession & Presto.CloseSession 446
APPENDIX E - PRESTO JAVASCRIPT API 448
Instance Methods 448
Constructor 448
AutoLogin 449
OpenSession 449
EndSession 450
SubmitRequest 450
PlayMacro 450
Window-Specific Methods 452
Setting and Getting the Window Title 452
Setting and Getting the Window Position 452
Setting and Getting the Window Size 452
Modifying and Working with Windows 453
OA-Specific Methods 454
SetFieldValue 454
SetSubfileFieldValue 454
Static Methods 454
GetConstant 454
GetField 455

APPENDIX F - INSIGHTS INTEGRATION 456
APPENDIX G - LICENSE KEY SERVER 464
License Server for Number of Users 464
License Server for Unlimited Seats 465
Presto Administrator Interface 465
Seat Count Utility 467
APPENDIX H - MDCMS CHANGE MANAGEMENT INTEGRATION 469
How to Configure MDCMS to work with Presto Change Management 469
How to use MDCMS in Presto 474
APPENDIX I - HEALTH CHECK TOOL 478
Object Status Reported 478
Non-Object Statuses 479
Running from the Command Line 479
Running the Tool via HTTP Request 480
APPENDIX J - FREQUENTLY ASKED QUESTIONS 482
Question and Answers 482
APPENDIX K - WEB PAGE DESIGN FOR GREEN SCREENERS 489
IBM i Screen Design vs. Web Page Design 489
HTML: The Language that Isn't 491
Writing Your First Web Page 492
Formatting Text - Simple Tags 495
Placing Images on Pages 498
Using Color in Your Pages 499
Navigating Using Links 500
Defining Tables 502
Interactive Pages - Using Forms and Forms Controls 507
Forms 507
Text Input Fields 508
Code that Affects the Entire Page 511
JavaScript 511
Style Sheets 512
Meta Tags 513
CSS (Cascading Style Sheets) 515
Why Use CSS? 515
Why You Should Use StyleSheets Now 515

Using Inline Style Attributes 516
Using Inline Stylesheets 516
Using Links to External Stylesheets 516
Stylesheet Media Types 517
Using External StyleSheets to Change the Entire Page's Appearance 517
Stylesheet Coding Rules 520
Using Comments 520
CSS Selectors 520
Element Selectors 520
Class Selectors 521
ID Selectors 522
Descendant Selectors 522
Examples of More Complex Selectors 523
Dynamic Pseudo-class Selectors 525
The CSS Box Model 526
'Block' versus 'Inline' Elements 526
The 'Cascade' in Cascading Style Sheets 526
Saving Style Sheet Specifications to External Files 528
CSS Best Practices 528
CSS Magic 530
How To Create Pseudo-Submit Buttons 531
Changing the Appearance of Forms Controls 534
Hyperlinks 535
The <DIV> and <SPAN> Tags 536

APPENDIX L - SOURCE CODE COPYRIGHTS AND ATTRIBUTIONS 537

Scintilla and SciTE 537
Firebug 537
Blowfish 538
Newtonsoft.Json 538
CRCpp 538
Apache License 539

Index

Section 1 - New Features and Upgrade Guide

CHAPTER 1 - NEW FEATURES LIST AND UPGRADE CONSIDERATIONS 22
Presto 10.0 New Features 23
Upgrade Considerations 23
Why You Should Start an Upgrade by Cloning your Current Environment 24
Upgrade StrPcCmd Utility 24
Page Options Inheritance 24
Changes to Page-level custom Page Options 25
Changes to Element Height and Width 26
Dark Mode 26
JQuery Update 26
Transformation Position Adjusted Flag 26
Apache Changes for Presto REST API 26
Apache Changes for Presto Server API 27
License Key Change for Version 8 and Higher 27
License Key Enforces Seat Count 28
New License Server 28
Mitigate Some Security Risks by Not Allowing Access to the Uploads Folder 29
ParameterVersion Updated 30
Exodus55 30
CHAPTER 2 - UPGRADING TO PRESTO 10.0 32
Upgrade Overview 33
Step 1: Download Presto and Transfer the IBM i Install Files 35
Step 2: Install the Server Components 36
Step 3: Set your CCSID in the HTTP Server Configuration 40
Step 4: Start the HTTP Server Instance 40

Step 5: Confirm your Server Installation 42
Step 6: Install the Presto Designer 42
Step 7: Create a Connection to your New Environment 43
Step 8: Make any Required Post-Installation Changes, and Test 45
Step 9: Promote the Upgraded Environment(s) to Production 46
Creating New Development Environments 47
Using WebFacing with Presto 48
More About the Presto IBM Apache HTTP Server Instance 48
Troubleshooting 49

CHAPTER 3 - PRESTO NEW FEATURES HISTORY 51

Presto 9.9 New Features 51
Presto 9.8 New Features 52
Presto 9.7 New Features 52
Presto 9.6 New Features 53
Presto 9.5 New Features 54
Presto 9.4 New Features 54
Presto 9.3 New Features 55
Presto 9.2 New Features 55
Presto 9.1 New Features 56
Presto 9.0 New Features 56
Presto 8.9 New Features 57
Presto 8.8 New Features 58
Presto 8.7 New Features 59
Presto 8.6 New Features 59
Presto 8.5 New Features 60
Presto 8.4 New Features 62

Chapter 1 - New Features List and Upgrade Considerations

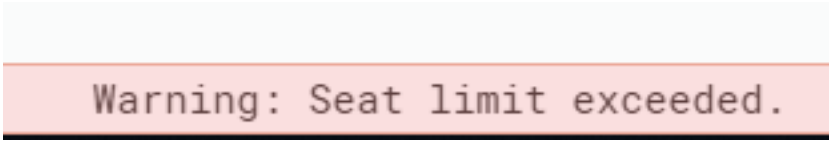
This chapter includes a summary list of new features in Presto 10.0 and a discussion of the changes you should particularly take into account when planning and performing your upgrade. Turn to Chapter 2 for the installation instructions.

License Key Enforces Seat Count
As of this version, Presto enforces the seat count that a license key is good for.

After the seat limit is exceeded, Presto will *not* allow additional users to log in.

If the seat limit is exceeded, any new Presto Administrator sessions will be notified with a message informing them:





Only Presto Administrators will see the above message.

You can determine the available seats and in-use seats of your license with the Presto Administrator Interface. See Knowledge Base article 16218 or section "License Key Server" on page 464 of Appendix 2.

Please call Technical Support if you have any questions or concerns!



This guide does NOT include upgrade considerations for previous releases.

The release previous to this one was Presto 9.9. If you are upgrading to Presto 10.0 directly from a release earlier than 9.9 you should review the upgrade considerations in the intervening New Features & Upgrade Guides. You can download & review earlier upgrade guides at the Presto documentation archives at myFrescheSolutions.com.

Presto 10.0 requires version 4 of the .NET environment. If you haven't installed it already, you may be prompted to upgrade when you install Presto 10.0 on your PC.

Presto 10.0 New Features

This section lists the highlights of this release. For more details on Presto's new features, please refer to our release notes in the Knowledge Base at myFrescheSolutions.com.

❑ **Health Check Tool**:

- **Active Subsystem.** The Health Check tool now checks for an active subsystem to launch PR_HUB jobs in. By default, PR_HUB jobs are launched in QUSRWRK, but the environment can be configured to use a custom subsystem (see https://www.myfreschesolutions.com/kb?kbnum=15438 for details). If a custom subsystem is configured, the Health Check tool will check whether that subsystem is active.

❑ **IDE Enhancements**:

- **jQuery.**Updated jQuery to 3.7.1
- **FusionCharts.** Updated FusionCharts to version 3.18.0

Deprecation Notice: As of Presto 10.1, the legacy communications middleware component, Exodus, officially deprecated in Presto 9.9, will be removed. Future Presto versions will no longer include Exodus. We strongly recommend customers currently relying on Exodus for client/server communication to transition to Presto's Server API. For any questions or assistance regarding the deprecation of Exodus, our support team is readily available to provide guidance and address concerns.

Upgrade Considerations

Presto is a two-pronged modernization solution:

- The "out of the box" prong of the software automatically analyzes, identifies and renders your 5250 data stream as web pages.
- The second prong to Presto is a toolset you can use to enhance the "out of the box" results. This requires that Presto correctly and consistently identify your screens every time the user visits them.

Every release includes improvements to both aspects of the tool. When you upgrade to a new version you should take time to review the following areas:

- How is this version of Presto handling screens you display to the user with no manual modifications (e.g. 'out of the box')?
- How is the new version handling your skin? How about the pages you have customized? Does everything look OK?
- Do any new tools or features allow you to improve your customization strategies?

Sometimes features in a new release may allow you to customize some pages to load more quickly, or apply simpler coding strategies.

Why You Should Start an Upgrade by Cloning your Current Environment

In the upgrade instructions you will see that our approach is to clone your current environment for evaluation and testing before you upgrade your production environment. There are a number of reasons for this:

- Presto supports a wide variety of page customization techniques. As we upgrade the software from version to version it is impossible for us to anticipate how every new feature will interact with your own approach to customizing your pages. Do not assume that all your custom changes will continue to work exactly the same after an upgrade.
- Every version improves the way Presto automatically recognizes and renders screen elements. In some cases these improvements can affect the underlying database link between the page and your custom changes. After creating a cloned upgrade environment, verify that your custom changes are still in place. If any have been dropped from a particular page or pages, re-attach them. In general we aim to avoid breaking links between pages and their customizations and known, unavoidable scenarios are documented in upgraders guides.

In short, always create and review a cloned environment prior to upgrading it to be the Presto environment that your users actively work in! The next sections include any additional, specific upgrade considerations for this release.

Upgrade StrPcCmd Utility

Updates to the StrPcCmd Utility will require a re-installation of the application. Presto includes the StrPcCmd Utility in your IDE installation folder. By default, it is installed to `C:\Program File(x86)\ESDI\Presto 10.0\StrPcCmd Installer.`

Double-click the installer executable file named `PrestoStrPCCmd_Installer` to initiate the re-installion on your PC. Restart the PrestoStrPcCmd program for the changes to take effect. Your current STRPCCMD configurations will be retained throughout the update process.

Please ensure that you perform these steps after completing the Presto upgrade to maintain compatibility and optimize performance.

Page Options Inheritance

In Presto 9.8, we've added Global Rules for Page Options. Previous to the addition of the Global Rules feature, when a custom page options file was applied to a screen, it would contain a clone of the page options as they appear in config.bws. This file would completely override all base environment settings. If the base config file were changed, those changes would not affect customized screens.

Page Options Inheritance was added alongside Global Rules in order to support the ability to layer the page options config, essentially creating a stack of configurations with the environment settings at the bottom, custom page-level changes on top, and any Global Rules that apply to the screen in the middle. Any config further up the stack can have any options set to “inherit” from further down in the stack, making it possible to only change values that need to be customized

while still allowing the environment config to be adjusted and have it apply to all screens which inherit those values.

Changes to Page-level custom Page Options

To support Page Options Inheritance, the existing Page Options tab of the Element Properties panel in the Visual Editor has been modified.

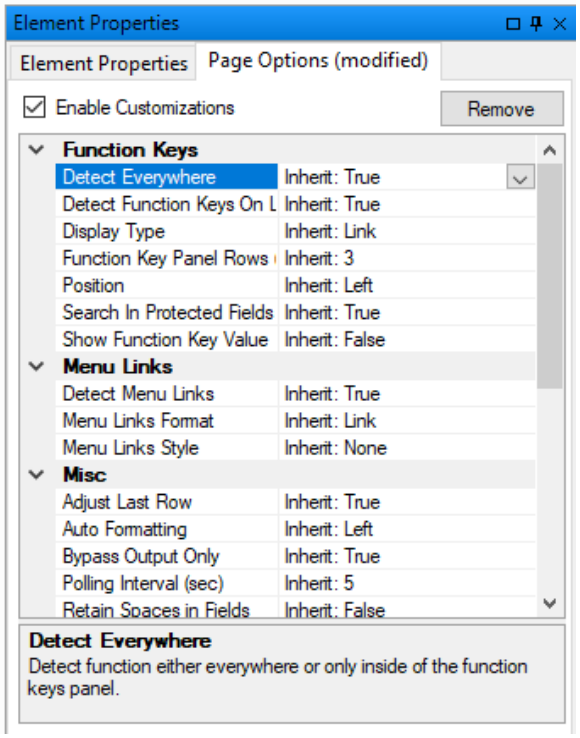


Figure 2-1. Page Options tab in the Visual Editor.

All properties now default to an **Inherit** state. Any properties that already had a drop down with options now have an additional **Inherit** option, with that value being selected by default.

The integer properties **Function Key Panel Rows** and **Polling Interval** have a drop down which will appear when you click in the field. The only option in the drop down is Inherit, which will apply the default. Clear the field to enter a new number.

The **Screen Title** property is unchanged from before. It is also blank by default. When blank, it will not override the default screen title. As there is no environment-wide equivalent, **Screen Title** has no **Inherit** option.

A further change to the page-level custom Page Options is that any values set to **Inherit** will not be saved to the resulting XML file. Previous versions would save all property values, regardless of their value. This change in behaviour is key to the function of Page Options Inheritance on the server. This also means that existing custom Page Options with all properties listed will continue to work as before, overriding all environment defaults with the stored values.

Changes to Element Height and Width

In Presto 9.5, we've added box-sizing to some elements, which impacts their width and height. Some element types (eg. input, auto-complete, text area, buttons, labels, passwords) may change size or shift slightly after upgrade. This adjustment could impact the alignment of elements after upgrade.

In previous releases, Presto only applied box-sizing to the default elements generated from screen fields, but not elements newly inserted by the Visual Editor. As a result, there could be a discrepancy between the size of default elements and inserted elements. As of this release, the elements are exactly the same size.

Dark Mode

In Presto 9.2, users can now switch to a Dark Mode.

Note the following:

- Dark mode may have issues with customized skins. In this case, dark mode will need to be disabled in the Advanced Configuration (config.bws). To disable dark mode, open config.bws and disable the 'ThemeSelector' module.
- Dark mode and theme switching is applied to the browser and stored in cookies. This means the theme is applied to all environments with the same server/domain.

JQuery Update

In Presto 8.8, we have updated the version of JQuery used by Presto to v3.51. This may impact some pages with custom Javascripts, as the new version will throw exceptions in some cases where the old version would silently fail.

Transformation Position Adjusted Flag

As of Presto 8.1, transformations have a new property called "Position is adjusted". In Presto 8.8, if this flag is not set, it will not attempt to shift the position of a transformation. For example, transformations that are inserted below tables. This means that in upgrades from before Presto 8.1, some transformations may need to be manually repositioned.

Apache Changes for Presto REST API

In Presto 8.7, we have introduced the Presto REST API. It is a REST-like API developed to allow programmatic interaction with the Presto Hub, from session creation to session end. As a result, Presto 8.7 and later include the following directives in the Apache HTTP server configuration:

```
# -- HubAPI Router

ScriptAliasMatch ^/presto(admin)?/([a-z]{2}/)?hubapi/.*$ /qsys.lib/QA_PR87RC5.lib/hubapi.pgm

<LocationMatch ^/presto(admin)?/([a-z]{2}/)?hubapi/.*$>
    # Rewrite output "text/json" to "application/json" so that Apache translates the
    outgoing data, but we have the correct output Content-Type

    Header edit Content-Type "text/json" "application/json"

    # Rewrite input "application/json" to "text/json" so that Apache translates the
```

```
    incoming data RequestHeader edit Content-Type "application/json" "text/json"

   Order allow,deny
   Allow from All
   AddHandler cgi-script .pgm
</LocationMatch>

RewriteRule ^/presto(admin)?/([a-z]{2}/)?hubapi/(.*)$ - [E=ROUTE:/$3]
```

Apache Changes for Presto Server API

In newer versions of Presto, we are making changes to how Presto communicates between the client and server components. As a result, Presto 8.5 and later include the following directives in the Apache HTTP server configuration:

```
# -- Presto API interface

ScriptAliasMatch ^/presto(admin)?/([a-z]{2}/)?api/.*$ /qsys.lib/XL_
PRESTO.lib/serverapi.pgm

<LocationMatch ^/presto(admin)?/([a-z]{2}/)?api/.*$>
       Order allow,deny
       Allow from All
       AddHandler cgi-script .pgm

       # Require a valid IBM i user, and switch the CGI job to run as them
       Require valid-user
       PasswdFile %%SYSTEM%%
       AuthType Basic
       AuthName "IBM Server"
       UserID %%CLIENT%%
</LocationMatch>

RewriteRule ^/presto(admin)?/([a-z]{2}/)?api/(.*)$ - [E=ROUTE:/$3]
```

These settings allow the Presto IDE to authenticate and communicate with the server via the API.

If you are copying custom changes from your existing Presto HTTP server instance, you will need to merge them with the above configuration settings replacing XL_PRESTO with the name of your Presto environment.



If you aren't familiar with editing an Apache HTTP server config, please see the section "Working With the Presto Apache HTTP Server Instance" in “Creating a New HTTP Server Instance” on page 395 of Appendix 2.

Please call Technical Support if you have any questions or concerns!

License Key Change for Version 8 and Higher

The license-key format was changed for Presto 8 and higher releases. This means that Presto requires a new license key. If you already own Presto and are upgrading to a new release, make sure that you have a new license key for version 8.0 and higher for all systems on which you run

Presto programs. Contact us at (250) 655-1766 or support@freschesolutions.com for details, or log in to myFrescheSolutions.com to get your new license keys.

How the New License Keys affect Backup, High Availability, or Disaster Recovery Systems

If you have setup a backup, high availability, or disaster recovery instance of Presto, you will need to consider that Presto 8.0 and higher license keys are now stored in the new data area `PW_407` in `XL_PRESTO`:

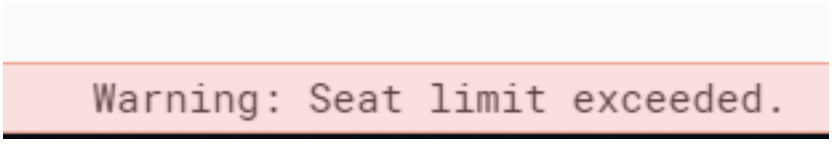
- If you have a Presto HA key, this rolls over to the new system, and you should be able to copy your complete Presto library to the new system.
- If you are doing the process manually, the license keys are stored in the new object `PW_407` in your Presto environment (`XL_PRESTO` by default). This data area should be replicated or omitted to your backup box depending on your manual HA process.
- Presto environments at version 7.3 or lower will still have their license keys validated with the previous data space `PW_406` in `XL_WEBSPT`. This data area will still need to be replicated or omitted to your backup box depending on your HA process and whether you have an HA key.

License Key Enforces Seat Count

Presto enforces the seat count that a license key is good for.

After the seat limit is exceeded, Presto will *not* allow additional users to log in.

If the seat limit is exceeded, any new Presto Administrator sessions will be notified with a message informing them:



Only Presto Administrators will see the above message.

You can determine the available seats and in-use seats of your license with the Presto Administrator Interface. See Knowledge Base article 16218 or section "Presto Administrator Interface" in “License Key Server” on page 464 of Appendix 2 for instructions.

Please call Technical Support if you have any questions or concerns!

New License Server

Presto license keys can be purchased for a restricted number of users or unlimited users. Presto now includes a new license server subsystem (introduced in version 8.1) to track the number of users.

Here are some additional technical details about this change:

- The license server is installed to a library named `XL_LCSRV`.
- The license-server subsystem is called `LICSRV`. This subsystem uses port 11111.
- The license server is installed and started during the install process.

- The Presto admin sessions will get a notification in the event that the license server is not active or there's a communication issue with the license server.

You can determine the available seats and in-use seats of your Presto License by using the Presto Administrator Interface. See Knowledge Base article 16218 or section "Presto Administrator Interface" in “License Key Server” on page 464 of Appendix 2 of the Presto User Guide for instructions.

If the seat limit is exceeded, any new Presto administrator sessions will be notified informing them of this:

```
Warning: Seat limit exceeded.
```

After the seat limit is exceeded, Presto will not allow an additional user to log on and output an error message.



Presto now enforces the seat count that a license key is good for.

If you’re starting EXODUS55 in a start-up CL program, you may want to change the STRSBS command to also include the new license server subsystem: `STRSBS XL_LCSRV/LICSRV`.

Please call Technical Support if you have any questions or concerns!

Mitigate Some Security Risks by Not Allowing Access to the Uploads Folder

One concern that has come up is Presto's /uploads/ folder. This folder is accessible to users through the Presto Apache server instance to support the signature pad. Users can potentially upload files, including HTML files with scripts, without signing into a valid session.

Block access to the upload folder by securing it in your Presto Apache HTTP server instance:

```
# Deny access to the "/uploads" folder
<LocationMatch ^/presto(admin)?/uploads>
        Order deny,allow
        Deny from All
</LocationMatch>
```



If you aren't familiar with editing an Apache HTTP server config, please see the section "Working With the Presto Apache HTTP Server Instance" in Chapter 2 for general instructions.

In a future version, we plan to add a configuration option to enable or disable uploads to the server.

Please call Technical Support if you have any questions or concerns!

ParameterVersion Updated

In newer versions of Presto, we have made significant improvements to how parameters are passed between screens. As a result, we introduced a new version of parameter passing (version 2).

However, we have an option in the Advanced Configuration file that allows you to force newer versions of Presto to use the older version of parameter passing.

The option is:

```
<ParametersVersion></ParametersVersion>
```

When you upgrade, Presto 10.0 now automatically updates this option to 2 to use the newest version of parameter passing.

We recommend that, under most circumstances, you leave this option at the default value 2. The older version of parameter passing is deprecated and, in the future, Presto may not support it.

This change affects elements that may have their values modified by custom scripts. After this upgrade, we recommend that you go to any screens that have custom scripts to verify that the customizations are still working.

Please call Technical Support if you have any questions or concerns!

Exodus55

Exodus is the communications middleware component used by the Presto Designer to transfer information back and forth to the IBM i. Presto 7.0 included a new version of Exodus.



These instructions were initially included in the Presto 7.0 Upgrade Guide and are repeated here in case you are upgrading to Presto 10.0 from a version earlier than 7.0.

Here are some additional technical details about this change:

- The new version of Exodus is installed to a library named `XL_EX55`.
- The Exodus subsystem is now called EXODUS55. This subsystem uses port 4015.
- The Exodus profile is now also EXODUS55.
- When EXODUS55 is installed, EXODUS51 will end and be disabled.

Eventually all Fresche products will be using the new Exodus.

If you're starting EXODUS in a start-up CL program, you'll have to change the STRSBS command to reflect the new subsystem: `STRSBS XL_EX55/EXODUS55`.

If you are running multiple versions of Presto concurrently, you can run the older versions of Presto with the new Exodus. The EXODUS55 subsystem is backwards compatible with products that were using EXODUS51. You cannot run the new version of Presto using the old Exodus.

The default port for Exodus is 4015. If you're using a different port, you can change the port for incoming connections to EXODUS55 by running the `XL_EX55/ZCFGTCP` command.

Chapter 2 - Upgrading to Presto 10.0

The instructions in this chapter will guide you through upgrading to Presto 10.0 from release 7.0 or higher. If you are currently at an earlier release, please contact Technical Support for additional instructions.

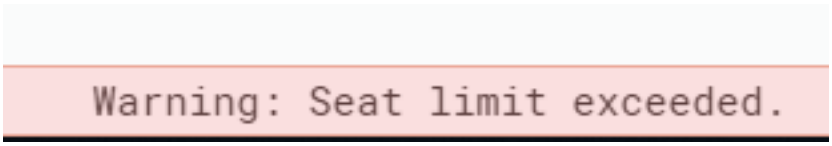


License Key Enforces Seat Count

As of this version, Presto enforces the seat count that a license key is good for.

After the seat limit is exceeded, Presto will *not* allow additional users to log in.

If the seat limit is exceeded, any new Presto Administrator sessions will be notified with a message informing them:



Only Presto Administrators will see the above message.

You can determine the available seats and in-use seats of your license with the Presto Administrator Interface. See Knowledge Base article 16218 or section "Presto Administrator Interface" in Appendix G of the Presto User Guide for instructions.

Please call Technical Support if you have any questions or concerns!



If you are installing Presto for the first time do not use these instructions. Please use the instructions on our web site or “Installation and Configuration” on page 7 of Appendix 2.

If you are upgrading to Presto 10.0 from a release earlier than 9.9 we strongly recommend that you review the upgrade considerations in all intervening Upgrade Guides (available in the Presto archives at myFrescheSolutions.com).

Upgrade Overview

You can use these Presto upgrade instructions to:

- Clone an existing environment at an older release to create a new test environment at the latest release (while leaving your current environment untouched).
- Clone an existing environment at your current release level (to create additional QA or test environments, for example).

The following sections include more details on these pathways.

New License Keys

The license-key format has been changed for Presto 8. This means that Presto requires a new license key. If you already own Presto and are upgrading to a new release, make sure that you have a new license key for version 8.0 for all systems on which you run Presto programs. Contact us at (250) 655-1766 or email at support@freschesolutions.com for details, or log in to myFrescheSolutions.com to get your new license keys.

How the New License Keys affect Backup, High Availability, or Disaster Recovery Systems



If you have setup a backup, high availability, or disaster recovery instance of Presto, you will need to consider that Presto 8.0 license keys are now stored in the new data area `PW_407` in `XL_PRESTO`:

- If you have a Presto HA key, this rolls over to the new system, and you should be able to copy your complete Presto library to the new system.
- If you are doing the process manually, the license keys are stored in the new object `PW_407` in your Presto environment (`XL_PRESTO` by default). This data area should be replicated or omitted to your backup box depending on your manual HA process.
- Presto environments at version 7.3 or lower will still have their license keys validated with the previous data space `PW_406` in `XL_WEBSPT`. This data area will still need to be replicated or omitted to your backup box depending on your HA process and whether you have an HA key.

What is Installed

Presto includes several components. The following components are relevant to an upgrade:

- **XL_PRESTO.** This is the default Presto product library.

  Note that if you have upgraded before or have multiple Presto installations your current Presto production library may have a different name, usually along the lines of XL_PRxx (where xx represents the 2-digit release level). The actual production library name has no bearing on how the software works. If for some reason the production library name does matters in your environment, you can use the command `RNMPRLIB` to rename your Presto product library back to a default name.



If you suspect you have multiple Presto environments but aren't sure what they are, run this command: `WRKOBJ OBJ(*ALL/PR_HUB) OBJTYPE(*PGM)`. The generated list will include all your Presto environment libraries.

- **XL_PRESTOI.** This is the Presto installation library.
- If you have a previous version of this library on your system the upgrade process will delete it for you automatically.
- **XL_PRDEMO.** An upgrade restores the demo library if it is not found on your system. If you want the latest version of this library delete your current instance of `XL_PRDEMO` before upgrading.
- **XL_EX55.** This is the Exodus55 Client/Server Communications Library.
- **XL_LCSRV.** This is the License Server Library.
- **IFS Folder.** An IFS folder on the IBM i named esdi/xl_presto by default, or matching the name of your product library.
- **Apache HTTP Server Instance.** Presto creates and uses an Apache HTTP server instance named `XL_PRESTO` by default. As with the IFS directory the name will normally match the environment library name you set on installing.
- **Presto Designer (PC Design Tool).** On the PC you will install the Presto Designer, the tool you use to customize your programs. If you have multiple, active Presto installations at different releases, you should also have multiple PC-side installations in different directories.

You can remove the library `XL_PRESTOI` from your system at any time after installing the software.

Recommended Upgrade Path

This is the recommended upgrade path, offering the least risk to your production applications:

1. Freeze development on your production Presto site.



Before proceeding with an upgrade make sure you have reviewed and understand all the upgrade considerations in the section "Upgrade Considerations" on page 23 of Appendix 2

2. Run the upgrade, selecting the option to create a clone of your production environment (with its own IFS directories, Apache HTTP server instance, etc.).

   Note that depending on the files or programs locked by your production application this step may require you to end your production HTTP server instance (while the cloning process runs, probably for 5-15 minutes). You can restart it immediately after the upgrade finishes running.

3. Test the cloned environment running separately on its own port.
4. When you finish testing, schedule a time to finish the upgrade when your users are offline. The last step takes only 1 or 2 minutes.
5. At the scheduled time end the Apache HTTP server instances for both the production and cloned environments. Then change the HTTP server instance for the new, tested, clone environment to use the port number previously used by the production environment.
6. Save and restart this server instance.

This approach represents the least amount of risk in the step going from your tested environment to production, as the version you put into production is exactly what you test and the only change is to the port number in the Apache server instance.

Keeping your Library Names Consistent

If you use this approach the name of your production library will change. If you need your production Presto library to retain its name after an upgrade you can use the command `RNMPRLIB` to rename the Presto library. The command is described in our online Knowledge Base Article 9436:

https://www.myfreschesolutions.com/kb?kbnum=9436

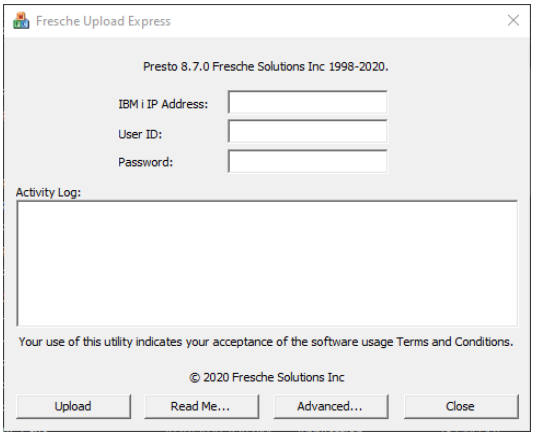
Please contact Technical Support if you have any questions about this process.

Step 1: Download Presto and Transfer the IBM i Install Files

Follow these steps to download the installation files and transfer the server component to your IBM i:

1. Log in to myFrescheSolutions.com to download the most current version. Download both the server and PC installation files.
2. When the downloads finish double-click the IBM i install executable file prestoserver-10.0.exe to unpack the files and begin transferring them to your IBM i.

   When the window below appears, enter your IBM i IP address or domain and user profile/password (`QSECOFR` is not required):



   The upload utility presented when you execute the IBM i install file.

- Click the Upload button. When the upload finishes the activity log displays the message 'File upload complete' and instructions on how to continue.



If the upload seems to be taking a long time, you can check for activity on the IBM i. The transfer job will typically be a job named QTFTPxxxxx in subsystem QSYSWRK, with a CMD-CRTSAVF function. If the job seems hung up, check the joblog.

When the file transfer finishes you can close the Fresche Upload Express window shown above.

Step 2: Install the Server Components

Follow these steps to upgrade the Presto server-side libraries.

The EXODUS55 Subsystem

The Presto Designer uses a communication middleware component to transfer data between the PC design tool and the IBM i. The component is also used by other Fresche products including WebSmart, Catapult, and Spool-Explorer/400.

- Until Presto 7.0, the communications middleware component was called EXODUS51 (in library XL_EX51).
- At release Presto 7.0 and higher, the communications middleware component is called EXODUS55 (in library XL_EX55).

Check for Active Exodus Jobs

To ensure that Exodus doesn't have locks on any objects that need to be replaced or upgraded, check the following before upgrading:

- At an IBM i session run the WRKACTJOB command and check whether the EXODUS55 or EXODUS51 subsystems are active. If the subsystems are not active, continue with the next section.
- If the EXODUS55 or EXODUS51 subsystems are active someone at your site may be working with an application using these. To see a list of these jobs run the commands WRKOBJLCK XL_EX55 *LIB and/or WRKOBJLCK XL_EX51 *LIB. Contact the owners of these jobs to request that they end their connections, or you can end them yourself.

After ending all EXODUS51 and/or EXODUS55 jobs you can use the ENDSBS command to end the subsystem(s). This will not affect any of your programs.



Note that EXODUS55 may be used by other Fresche products, including Catapult, Spool-Explorer and WebSmart.

Check for Object Locks

To ensure that the upgrade proceeds as smoothly as possible log on to your IBM i as QSECOFR and run the following command to check for object locks:

```
WRKOBJLCK OBJ(XL_PRESTO) OBJTYPE(*LIB)
```

If your active environment library is named something other than XL_PRESTO, use its name when running this command.

After verifying that all the object locks are cleared, continue with the next step.

Locks from Your Apache HTTP Server Instance

If your production library is locked by your Apache HTTP server instance you can run the following command to end the XL_PRESTO HTTP server instance AFTER confirming that you don't have any users currently on the system:

```
ENDTCPSVR SERVER(*HTTP) HTTPSVR(XL_PRESTO)
```

If the Apache instance for your production environment has a name other than XL_PRESTO, specify that name instead. All the jobs should end within a few seconds. Then check again for object locks. You can restart the server instance as soon as the upgrade completes (5-15 minutes).



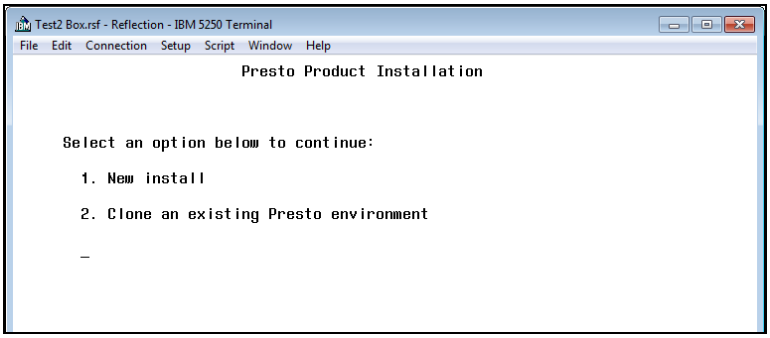
You can also end the HTTP server instance from the online interface, at http://youripaddress:2001/HTTPAdmin. If you aren't sure what the server instance name is, this is a good way to do it.
From the Admin interface, you need to select the Presto Apache HTTP server name from the Server drop down list, then click the Stop button.

Then continue with the instructions on the next page.

Install the Server Components

Follow the steps below to begin installing the Presto server components.

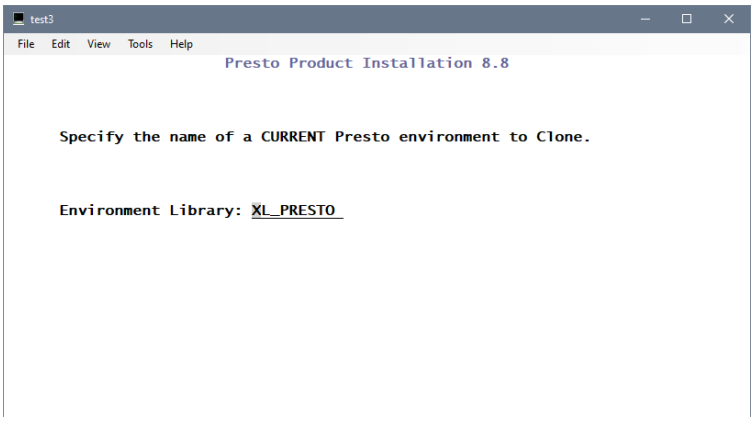
1. Execute the command XL_PRESTOI/BCDINSTALL.
2. This prompt is presented first:



   Choose to install a new environment or clone/upgrade an existing one.

   - Select option 2 to clone and upgrade an existing Presto environment, then press Enter.

3. When prompted, specify the name of a Presto environment to upgrade:



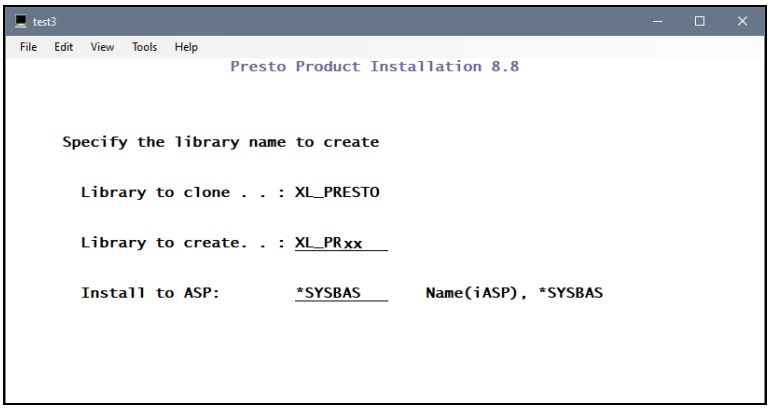
Selecting the environment to upgrade. The default is XL_PRESTO.

- If this is your first upgrade, the library to upgrade is probably `XL_PRESTO`. If you have upgraded multiple times, the library to upgrade may have the name `XL_PRxx`, where 'xx' corresponds to the previous release level (e.g. '60' for version 6.0).
- Press Enter to continue.

The upgrade will present additional options before performing any actions on this library.

---

4. At the next panel select the new environment name. You may also enter an iASP name if you opt to install on an ASP:



Upgrade options.

- Presto defaults to naming your cloned environment `XL_PRxxx`, where 'xxx' corresponds to the current release level.
- The install will create three objects with the new name:

- A new Presto environment library, `XL_PRxxx`. Your existing environment is not changed in any way.
- A new IFS directory (/esdi/xl_prxxx) with copies of all your skins and custom pages.
- A new HTTP server instance, `XL_PRxxx`.

- Note that this HTTP server instance is NOT a clone of your existing Apache instance. It is a completely new instance with the paths & libraries pointing to the cloned environment. If you have customized your existing instance you may need to apply these changes to the new instance manually as well.

Press Enter above to proceed to the next step of confirming the new IFS file location

When the upgrade is complete it presents instructions for configuring and starting your HTTP server instance:

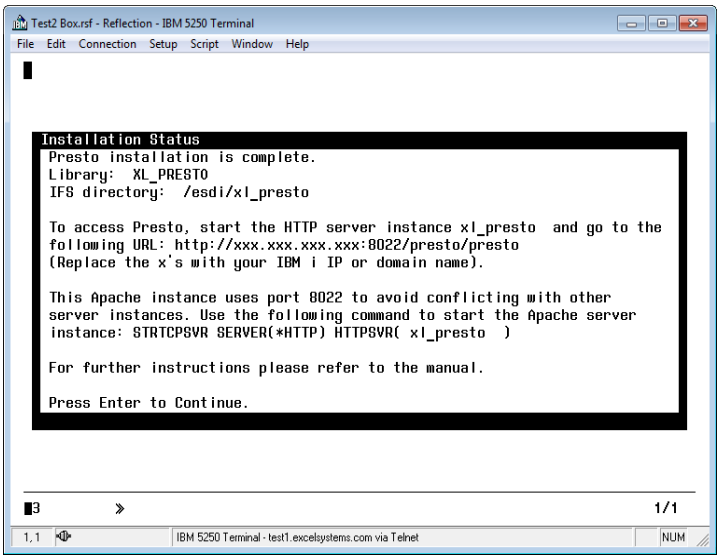


Figure 2-1. Post-installation instructions (the release level may vary).

The Apache HTTP server instance for the cloned environment will default to using port 8023 (a new install uses port 8022).



If this is your third (or subsequent) Presto environment you may need to manually change this default port number before starting the server instance.

Continue with the next step below.

Step 3: Set your CCSID in the HTTP Server Configuration

This step is only relevant if your system has a CCSID other than 37 AND you have created a cloned copy of your production Presto environment.

When you upgrade from a previous version the Presto Designer (the PC component) will pick up your CCSID settings from your previous installation. On the IBM i you need to adjust the new, cloned HTTP server instance manually.

Verify your system CCSID by running the command `DSPSYSVAL SYSVAL(QCHRID)`. If the value displayed for the Code Page is 37, you can continue with the next step. Otherwise this will be the value you will use in your Exodus55 configuration for the DefaultFSCCSID, as described below.



Note that the settings in this step are ONLY required if your CCSID is not 37. For a CCSID of 37 (United States/North America) the default configuration will work for you.
If you aren't sure where or how to make these changes, review the instructions in the section Chapter 2 at the end of this chapter.

Change the DefaultFSCCSID and DefaultNetCCSID values.

Go to your configuration and find this line:

```
DefaultFsCCSID  00037
```

Change it to:

```
DefaultFsCCSID  00xxx
```

Replace xxx with your system's CCSID. Save your changes.

In the next step you will restart the Apache instance.

Step 4: Start the HTTP Server Instance

If in Chapter 2of this upgrade you selected the option to create a cloned environment (Chapter 2) the installation created a new Apache HTTP server instance with the same name as your upgrade library. The last screen of the clone server install (Chapter 2) confirms these details for you and provides instructions for starting that HTTP server instance.

If you upgraded your production library directly, your existing HTTP server instance is not touched by the upgrade process.



A cloned Apache HTTP server instance is always initially configured to use port 8023. If you are upgrading more than one environment you may need to manually change the port number used by new instances as required.

Re-Apply Any Custom Changes

If you had previously made any custom changes to your Presto HTTP server instance, apply these changes to the newly installed server instance.

Start the New HTTP Server Instance

If you haven't already, run this command to start the server instance:

```
STRTCPSVR SERVER(*HTTP) HTTPSVR(XL_PR100)
```

If you used a different upgrade library name, replace `XL_PR100` with that name. After a few seconds, you can run this command to confirm that the HTTP server was started:

```
WRKSBSJOB SBS(QHTTPSVR)
```

You should see several jobs with your environment name in this subsystem.

See the section Chapter 2 for further details.

Confirm the EXODUS55 Subsystem is Running

This is also a good time to confirm that the `EXODUS55` subsystem is running. Unless you explicitly ended it before the upgrade it should still be active, but you can verify this by running the `WRKACTJOB` command.

You should see the `EXODUS55` subsystem with a job named `EXSERV` below it. If not, start the subsystem now by running this command:

```
STRSBS XL_EX55/EXODUS55
```



Port 4015. The default `EXODUS55` port is 4015. If you're using a different port, you can change the default port of incoming connections using the `XL_EX55/ZCFGTCP` command (press F4 to prompt the command, and press Enter).
If you change this port number, restart the `EXODUS55` subsystem.

Confirm the LICSRV Subsystem is Running

Confirm that the `LICSRV` subsystem is running. Like the `EXODUS55` subsystem, it should still be active after the upgrade, but you can verify by running the `WRKACTJOB` command.

You should see the `LICSRV` subsystem with a job named `LICSRV` below it. If not, start the

```
STRSBS XL_LCSRV/LICSRV
```



The upgrade replaces the version of the license server. Your existing connections might get the error: `Error: There was an error fetching the status information. Please contact status information. Message: Server denied SetProduct`
To not affect existing connections if the license server was running during the upgrade, you can manually end and restart the `LICSRV` subsystem. Run this command to end the subsystem:

```
ENDSBS LICSRV
```

Then run this command:

```
STRSBS XL_LCSRV/LICSRV
```



The LICSRV PORT. The default `LICSRV` port is 11111. You can change the default port of incoming connections by changing the value of the `XL_LCSRV/LICSRV` data area.

Then continue with the next step.

Step 5: Confirm your Server Installation

Now you're ready to confirm your server side installation. To begin using Presto, open a browser session and enter this URL:

```
http://your_domain:port/presto/presto
```

Insert your domain name and the port for your new HTTP server instance. (for instance 'intra.company.com:8023' or '192.168.0.100:8023').

With this URL, you should see a sign-on screen in your browser window, matching the look of your current site.



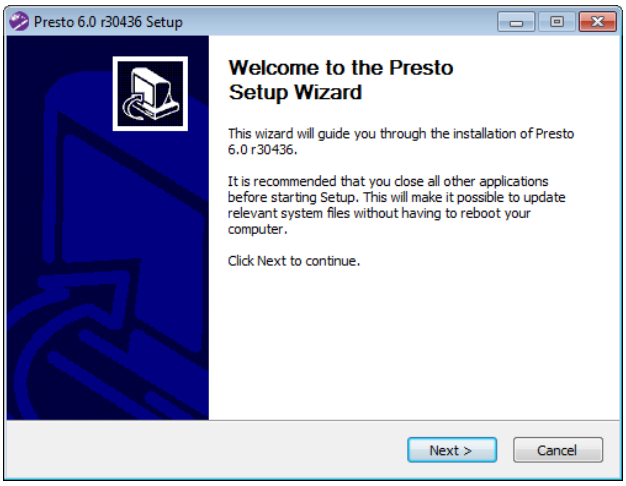
Remember that Presto defaults to using port 8023 for every upgrade environment, so if this is your third (or higher) upgrade and you changed the port in Chapter 2, use that number here.
If you are installing to a new system and need a new license key, please contact our sales office. You can find license key entry instructions at our web site: http://www.bcdsoftware.com/iseriessupport/codes/.

Now continue with the instructions on the next page to install the PC components.

Step 6: Install the Presto Designer

Follow the steps below to install the Presto PC design tool (the Designer):

1. If you haven't yet done so, download the latest version of the Presto client from myFrescheSolutions.com. The file is named PrestoClient-10.0.exe
2. When the download finishes, double-click the downloaded file to unpack the install files and initiate the install:



Installing the Presto PC components.

Your exact release level may vary from the one illustrated here.

3. Follow the instructions to install the Presto executable to your PC.



If you are creating a clone of your current production environment, when the installation prompts you for the path to the new executable add \10.0\ to the end of the path (or another unique value) so you don't overwrite your currently installed Presto Designer. You need your currently installed version to work with your existing environment.

4. Click Finish when the installation completes.
5. The install will give you the option to launch the software immediately - do this.

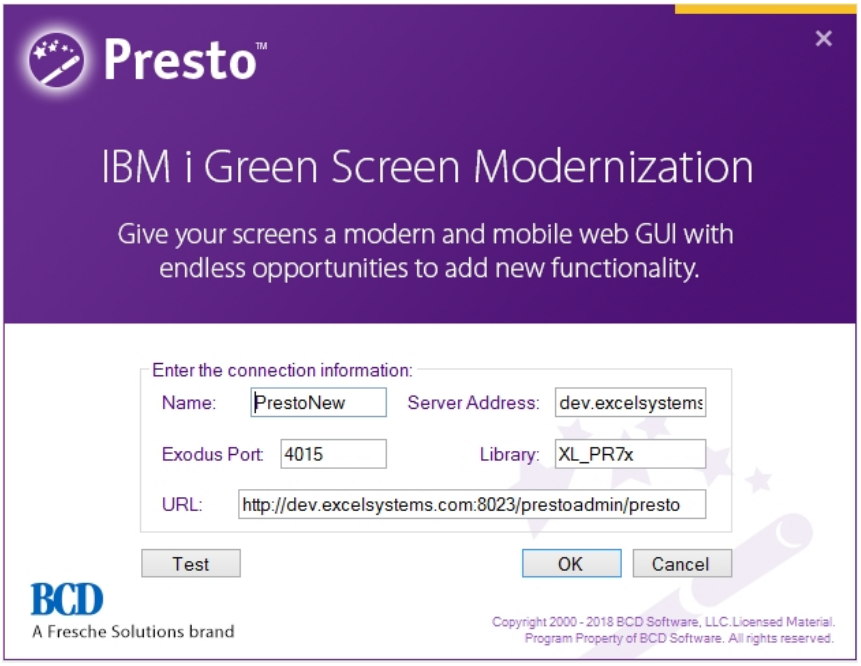
If you are installing a second instance of the Presto Designer to a non-default location remember to also add a shortcut on your desktop for this new version.

Step 7: Create a Connection to your New Environment

You must configure the Presto Designer to connect to your new environment on the IBM i so you can test the upgrade.

1. When you complete the install Presto prompts you to launch the software. Go ahead and do this, but when prompted to connect to the server, click the Cancel button on the Login prompt.

2. In the 10.0 Presto Designer open the Tools/Connections menu option. You should see a list of your existing environments.
3. To test your new environment you need to add a connection to this list. Click the Add button to present this window:



Configuring a Presto connection.

- Enter the values as described below:

Field	Value
Name	Enter a descriptive value for this connection.
Server Address & Port	You can use your existing server address and port number (the default is 4015).
Library	Type the name of your upgrade library.
URL	This value should be http://yoursite:xxxx/prestoadmin/presto, where 'yoursite' will your server address (which Presto inserts by default), and xxxx will be the port you are using for your upgrade HTTP server instance.

4. After entering the required values use the Test button to verify your settings. Sign on to your IBM i when prompted.

When you click OK Presto should confirm that you have connected.

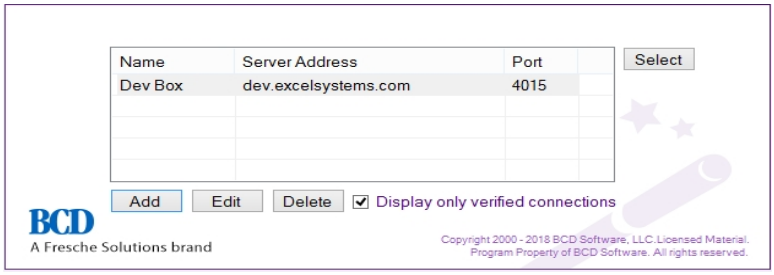
Here are a couple of things to check if you don't get a successful connection:

❑**The EXODUS55 subsystem is active.** The most common connection error (10061) is caused by the EXODUS55 subsystem not being active. If you get this error, run the WRKACTJOB command and check whether the EXODUS55 subsystem is running and contains a job called EXSERV. If the subsystem isn't started, run the command STRSBS XL_EX55/EXODUS55. If the subsystem is running but EXSERV is not, end and restart the subsystem and confirm the job is added. Then try connecting again.

❑**Other connection errors.** If the subsystem is up but you can't connect, add XL_EX55 to the library list of your IBM i session and run the command ZDSPERRLOG to display a list of your connection attempts. Check for error messages (such as invalid User IDs, etc.).

If you can't spot the problem immediately please call Technical Support at (250) 655-1766, or email us at support@freschesolutions.com.

5. After Presto verifies that your connection is working, click OK to save the connection and return to your list of connections. Click OK again at the list of connections to return to the Designer.
6. You should now see your new connection in the list at the top:



The new connection in the Designer (exact name may vary)

7. At this time the Designer will also automatically prompt you to connect to your system. Sign on to your system.

Now continue with the upgrade instructions on the following pages which describe further configuration changes you need to implement for this release.

Step 8: Make any Required Post-Installation Changes, and Test

Step 1: Review and Apply Any Relevant Post-Installation Changes

After you complete the upgrade steps above review the items in the section "Upgrade Considerations" on page 23 of Chapter 2. There are certain types of changes that we cannot automatically do during an upgrade because they are related to files that you may have customized or changed yourself.

For example, an upgrade will never directly change your Presto Apache HTTP server instance. Any changes required will need to be made manually, and the server instance restarted at an appropriate time.

Step 2: Confirm your Critical Programs Work Correctly

If you followed the recommended upgrade path to create a cloned environment, now is the time to test it.

Start by making sure your most critical applications continue to behave as you expect. If you have a core set of screens that your users are frequently using, start by working your way through them, ideally using some test data.

If you have a small group of particularly competent users you can also ask them to help you with this process; people used to seeing the screens regularly will be more likely to notice changes in appearance or behavior.

Step 3: Review Custom Pages and Pages Edited with the Visual Editor.

Some upgrades require changes that affect the internal database files that link your screens to the customizations you have created for them.

You should verify that any customized screens still have all the customizations, and they continue to work as expected.

The section “Upgrade Considerations” on page 23 of Chapter 2 describes the changes that we anticipate in this version.

Step 4: Review your custom.js file and any datepicker localization changes.

The datepicker you can implement via the Presto visual editor matches the typical settings for an English speaking audience. If you are building your application for another country you can take advantage of a jQuery data localization feature. To check this open your custom.js file (using the Presto Designer menu option Tools>User Resources>custom.js). The default file includes instruction for switching the language settings.

Step 5: Autologins

If you are using an autologin you must copy the data area `PRLIBL` from your old Presto library to XL_PR100 (your new Presto library) and update the library list to use your new Presto library instead of the old. See the User Guide for details.

Step 9: Promote the Upgraded Environment(s) to Production

After fully testing your upgrade you can promote those changes to production. The section “Chapter 2” on page Chapter 2 discusses your options.

Step 1: Change the New HTTP Server Port Number

End the HTTP server instance created by the upgrade process and edit the configuration. See the section “Chapter 2” on page Chapter 2, below, for additional information.

In the HTTP server instance configuration, change the port to the port number currently used by your active site.

Do NOT start this server instance yet!

Step 2: End the Production Server Instance

At a time when your users do not require access to your application end the production Presto server instance. You can use this command:

```
ENDTCPSVR SERVER(*HTTP) HTTPSVR(XL_PRESTO)
```

If your production Presto HTTP server instance is not named XL_PRESTO, then reference your actual instance name.



You can also end the HTTP server instance from the online interface, at http://youripaddress:2001/HTTPAdmin. If you aren't sure what the server instance name is, this is a good way to do it. From the Admin interface you need to select the Presto Apache HTTP server name from the Server drop down list, then click the Stop button.

All the jobs should end within a few seconds.

Step 3: Start the Upgraded Server Instance

Now start the upgraded HTTP server instance:

```
STRTCPSVR SERVER(*HTTP) HTTPSVR(XL_PR100)
```

Reference your actual upgrade library on this command, if not XL_PR100.

At this point your users should be able to continue using your applications.



These steps assume that you are using Presto's default HTTP server configuration. If you have made custom changes, or merged server instances, you need to take those changes into account on upgrade.

This concludes the upgrade instructions for Presto 10.0.

Creating New Development Environments

A side-benefit of the upgrade process outlined in the previous steps is that it is easy to create development environments in which to work without affecting the users running on the live site.

To create a development environment, re-run the following steps:

- ❑ "Step 2: Install the Server Components" on page 36
- ❑ "Step 4: Start the HTTP Server Instance " on page 40.
- ❑ "Step 7: Create a Connection to your New Environment" on page 43.

You just need to make sure that when you create a new HTTP server instance for your development environment that you configure it to use a different port.

Finally, you need to send your development team the library name and port number for the new dev environment.

If you have any questions about any part of this process, please get in touch with our Technical Support team.

Using WebFacing with Presto

If you are using WebFacing with Presto you can find configuration instructions in the User Guide.

More About the Presto IBM Apache HTTP Server Instance

The installation process is designed to automatically create and start an Apache web server instance called `XL_PRESTO`. To confirm that this server instance was successfully started, run the `WRKACTJOB` command and look for the subsystem `QHTTPSVR`. This subsystem should contain several jobs named `XL_PRESTO`.



If this server instance is not visible you can start it using the command `STRTCPSVR SERVER(*HTTP) HTTPSVR(XL_PRESTO)`. If you have any difficulty with this process, contact Technical Support.

The default configuration will normally get you up and running, though you may decide to make changes to it before promoting applications to production.

Working with the Presto HTTP Server Instance

As stated above, the default configuration on install should work for you, but should you need to, you can make changes to the `XL_PRESTO` HTTP Server configuration via IBM's Web Administration for iSeries pages. You can access this interface from your browser, at http://your_server_ip:2001/HTTPAdmin (insert your iSeries IP address before the :2001). Log in using your iSeries user profile.

The page presented includes a list of your HTTP servers in a dropdown box. Find and select the Presto entry (as shown below):

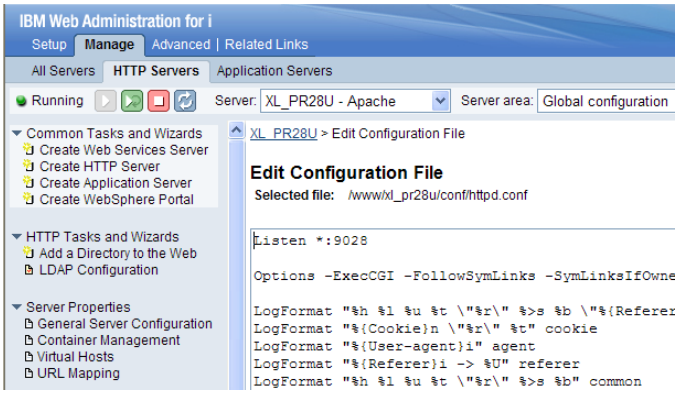


Figure 2-2. Working with your iSeries HTTP servers via the online interface.

If the server isn't running (i.e. the status is 'Stopped'), you can click the green button to start the server instance, to change the status to 'Running'.

Changing the Presto Server Instance Default Port Number

To avoid the possibility of conflicting with any existing HTTP server instances you may have running on your system, we ship the XL_PRESTO server instance configured to use port 8022. If you already have something running on that port, you can change this value in the configuration file.



If you aren't running any web applications on your IBM i, you can also run your Presto applications on port 80, which removes the need to include a port number in the URL.

If you have any questions about how to change your HTTP server configuration, or whether you need to use a different port number, please contact Fresche Technical Support.

Troubleshooting

Here are a few troubleshooting suggestions specifically related to upgrades. The User Guide includes additional items.

Errors Processing JSON Response when Launching the Presto IDE after Upgrading

If on launching the Presto Designer after an upgrade you get a Error Processing JSON Response, confirm that there are references in the web server's configuration file for the communication via API between the client and iSeries components. For the configuration directives, see the section Upgrade Considerations in Chapter 2.

JavaScript Mismatch Errors when Launching the Presto IDE after Upgrading

If on launching the Presto Designer after an upgrade you get a JavaScript version mismatch error, first confirm that both the client and iSeries components you are using together are at the same release level.

If they are then, continue with checking the items below.

HTTP Server IFS References

Check that the library (ScriptAliasMatch) and directory (AliasMatch) references in the web server's configuration file are pointing to the correct IFS directory and library.

The configuration file, httpd.conf, is located in the www/ENV_NAME/conf/ directory on the IFS (where ENV_NAME is the name of the environment you installed to). The library and directory references should match the name of the web server. So, for example, if you installed into the XL_

PR100 environment then the configuration file will reside in www/XL_PR100/conf/ and the library and IFS directory references should be to XL_PR100.

HTTP Server Config File Reference

If you still encounter the error message after making the above change and restarting the web server, the next step is to verify that the member entry for the web server in the QUSRSYS/QATMHINSTC is referencing the correct config file.

To check this, run the command STRPDM then take an option 3. The file is QATMHINSTC in library QUSRSYS, and the member will have the same name as the web server (XL_PR680 for example, using the environment mentioned above). Take a 5 to view the contents of the member and verify that it's pointing to the correct configuration file. If not, change it then save the file and restart the web server.

Chapter 3 - Presto New Features History

This chapter summarizes the most notable new features in previous versions of Presto. For new features in the latest release, see “New Features List and Upgrade Considerations” on page 22 of Appendix 2. For further details, you can download & review earlier upgrade guides at the Presto documentation archives at myFrescheSolutions.com or refer to our release notes in the Knowledge Base.

Presto 9.9 New Features

This section lists the highlights of this release. For more details on Presto's new features, please refer to our release notes in the Knowledge Base at myFrescheSolutions.com.

❑License Server:

- The Health Check tool will now check whether the License Server is available, and whether licenses are invalid or expired. For licenses with unlimited seats, the Health Check tool will not respond with an error if the license server is inaccessible.
- Added additional logging on the license server connection for ease of debugging.
- Timestamp added to License Server file names.
- Added 10 second wait for license server responses to reduce the likeliness of Presto timeouts.

❑Change Management:

- New option in Settings to scrub exported data. Presto will exclude variable data from the exported XML (e.g., timestamp, sequence ids). By including this type of data, tools like SVN and Git will report diffs when files were exported from different environments.
- Presto now allows you to specify the path for Import backups in config.bws. This is useful for users with read-only authority on Presto IFS directories.
- Added error logging when an Import fails to create the backup.

❑IDE Enhancements:

- **Subfile Options.** A new advanced config javascript option <SubfileOptionTrigger> allows you to set the behaviour of the subfile flyouts. They can either be to set to display only when clicking on the row, or only on hovering over the input field.

- **Signature pad.** The signature pad plugin was updated for improved support on touch screens.

❑**Security:** Improved security by updating Object and IFS file permissions on server installation, in accordance with Trinity Guard best practices.

Deprecation Notice: As part of our commitment to providing cutting-edge and streamlined services, we would like to inform users of the deprecation of Exodus. In an upcoming Presto 10+ release, Exodus will no longer be actively supported or maintained. Customers who currently rely on Exodus for client/server communication are encouraged to transition to using Presto's Server API. For any questions or assistance regarding the deprecation of Exodus, our support team is available to provide guidance and address concerns. Contact support@freschesolutions.com for prompt assistance.

Presto 9.8 New Features

❑**Global Rules for Page Options**: Presto provides a new system which allows the user to create a set of general rules for identifying screens and customizing the page options of those screens. It is intended for applying the same custom page options to groups of similar screens, rather than the existing systems which are designed for uniquely identifying screens.

For example, you may want to update all Menu screens to disable table detection or set the menu links to display as buttons. Assuming your Menu screens share certain identifiable characteristics, you should be able to create a rule that will match them all and apply the options changes automatically.

❑**Server Installation**:

- **Uninstaller.** Presto now ships with a program to uninstall Presto environments.
- **IASP.** Presto will automatically set the ASP Group when installing new environments on an IASP, reducing potential install errors.

❑**IDE Enhancements**:

- **Format.** New date picker field format of dd/mm/yy.
- **JQuery UI.** Updated JQuery UI to version 1.13.2.

❑**Security Enhancements.**

- **Authentication.** Improved session handling at authentication to prevent session fixation attacks.
- **HTTP Headers.** Added security HTTP Headers Strict-Transport-Security, X-Content-Type-Options, and Content Security Policy. For more information on Content Security Policy, see https://content-security-policy.com/

❑**Spool File Viewer Timeout.** Presto now has the ability to specify a spool file generation timeout. The new SpoolFileTimeout environment setting controls how long Presto will wait for a Spool file to be returned by the server, allowing the export of large files.

❑**System Requirements:** Presto requires a minimum IBM i operating system of V7R3 for this release.

Presto 9.7 New Features

❑**Enhancements to the IDE**:

- Added fallback support for the IDE when External Internet is not available to access Google Fonts API.
- Improved ServerApi and HubApi error logging for connection issues.
- Improved support for Chrome in MacOS.

❑**Identities Handling**:

- Improvements in the handling of identities in custom windows.

❑**License renewal enhancement.** The grace period for renewing Presto licenses has been increased, allowing you to enter a new license one month prior to its expiration date.

❑**Security Enhancement.** Improved handling on "task=listsplf" parameter to prevent potential XSS attacks.

Additional Notes:

- On upgrade, Server API will default to **ON**. The IDE will display a confirmation dialog on attempts to uncheck Use API checkbox in the hidden settings.
- **Deprecation Notice:**
  - Presto 9.7 will be the last version to support **V7R1.**
  - The next release will no longer support **RPGOA.**

Presto 9.6 New Features

❑ **Health Check Tool.** Presto includes a new tool for checking the health status of an environment. The Health Check tool reports on any damaged objects in the library, checks for orphaned messages in the data queue, and checks on whether the data queue is nearing capacity. For more details, see “Health Check Tool” on page 478 of Appendix 2

❑ **Enhancements to the IDE**:

- **Restart Session button.** Disconnected screens will now include a restart button, prompting users to take action when a connection has been lost.
- **Retain spaces in fields.** The setting that determines how to handle text fields is now available as a page level option.
- **Field Formats test utility.** Presto now comes shipped with a Field Format test utility, available under PT_MENU. This utility can be useful for testing Presto behaviour with numeric fields, field plus/minus behaviour, etc.

❑ **STRPCCCMD.** Presto has added logging for the STRPCCMD utility. Logging is enabled via the Settings tab of the StrPcCmd Utility. Logs are written to C:\Users\[User]\AppData\Roaming\ESDI\Presto\StrPcCmd Logs\PrestoStrPcCmd-[current date].log. For more information, see “Access PC Applications from Your Presto Site” on page 99 of Appendix 2.

❑ **Ignore damaged data queues.** In environments with multiple data queues, Presto will now skip damaged data queues when starting up new sessions.

❑ **New Setting "Treat as Last Row As".** This global setting allows you to determine whether the last row should be treated as a regular row (raw text) or an Error Message field/subfile.

Presto 9.5 New Features

❑ **Enhancements to Visual Editor**:

- **Custom Themes.** Presto now includes a custom theme per Preset Color (dark, green, default, etc) in each Skin. See "Customizing Skin Themes" on page 51 of Chapter 2.
- **Presto iOS Scrolling.** TableTouchActionVersion is a new javascript option in config.bws for screen widths > 880px that allows scrolling of subfiles without scrolling the screen.

❑ **HubAPI**:

- **Screens API.** AutoFormat is a new ScreensAPI parameter that allows the disabling of automatic formatting on raw screen data returned from HubAPI.
- **Logging.** HubAPI now reads the logging level from config.bws. Levels all/info/timing will enable logging for HubAPI.

❑ **API Timeouts.** There is a new IDE Setting "API Timeout" that allows you to configure the maximum wait time for http requests. The default timeout is set to 5 minutes, to allow for long running exports. See "API Timeout" on page 49 of Chapter 2.

- **Apache Change for API Timeouts.** The `TimeOut` value in the Presto Apache Configuration has been increased from 300 (5 minutes) to 1800 (30 minutes):

```
# Define any and all IP addresses on the host:
TimeOut 1800
ListenBacklog 511
KeepAliveTimeout 300
MaxKeepAliveRequests 100
```

  This increased timeout supports exporting a large number of screens along with the new IDE API timeout configuration.

Presto 9.4 New Features

❑ **Visual Editor Enhancements**:

- **Function Key Options.** Presto now has the ability to add, hide, and edit function keys in the Visual Editor. The properties for individual function keys can easily be changed with the full spectrum of options available for Link type elements. See "Function Key" on page 215 of Chapter 2.
- **De-select elements.** You can now de-select elements by clicking on empty space in the Visual Editor.
- **Disable auto-align.** Temporarily disable auto-align guidelines by pressing Ctrl while dragging an element.
- **Line height.** The option to change the line-height has been added to Text settings.
- Improved performance when saving Visual Editor changes.

❑ **IDE Enhancements**:

- **Title bar details.** The environment name and host name are now included in the title bar, making it easy to see the connection details of the current environment.

- **PageHeader controls.** Presto now has the ability to enable/disable PageHeader via Advanced Settings (config.bws).
- **Menulinks.** Added the ability to set the maximum column distance between the number and option name for menulinks in config.bws.
- Updated Exodus to use .Net Framework 4.8

❑ **Spoolfile Viewer**:

- New clickable icons for "View as Html" and "View as PDF", with tooltips to describe the action.
- Display deleted filename in confirmation popup.
- Rows are highlighted while hovering.

❑ **Logging.** Changed the naming format of server log files to have the date at the beginning of the filename, and renamed PrestoAPI to ServerAPI..

❑ **Macros.** Support for the Attention (ATTN) key within macros.

❑ **Skin Editor.** Fonts can now be changed via the Skin Editor. New fonts have been added to the font-family options, which are also available in the Visual Editor.

❑ **Improvements for Kerberos:**

- Added Kerberos configuration in the shipped Apache configuration file (httpd.conf). This is commented out by default.
- Allow one Apache configuration to serve Kerberos to regular users, and non-Kerberos users for the Presto IDE. Previously you needed to set up two different Apache web servers - one for Kerberos and one without.
- Increased performance of Kerberos-enabled setups by halving the number of web requests required (reduced latency by half).

Presto 9.3 New Features

❑ **Enhancement to Visual Editor's Responsive Grid.** Elements in a Responsive Grid cell will be highlighted when other elements are dragged over them, indicating where the dragged element will be dropped so the user can better see where the element will be inserted.

❑ **Menulinks.** The menulinks default from the carat character has been changed to numbers.

❑ **Logging.** Added logging to HubAPI controllers, and improved IDE client side and server-side logging for Server API. These will ease debugging.

Presto 9.2 New Features

❑ **Dark Mode.** Users can now toggle to Dark mode on a Presto session (new Midnight theme). This theme switching feature is configurable and allows for a selection of custom themes.

❑ **Visual Editor Enhancements**:

- **Arrangement Tools.** Presto now gives you a new suite of arrangement tools. These tools can be found on the right click menu, or on the Toolbar:

- Align elements to the Left, Right, Top, or Bottom
- Distribute elements Horizontally or Vertically
- Bring to Front and Send to Back

- **Element Alignment Guidelines.** The Visual Editor now displays colorful guidelines while dragging or resizing elements to indicate when when they align with the center or edges of other elements.
- **PNG File Support.** The Visual Editor now supports the use of PNG files for image elements.

❑ **IDE Integrated Javascript Debugger.** The WebBrowser control has been replaced with WebView2, to support modern web features and debugging tools.

❑ **IDE Logging.** When the IDE does a version check with the server to ensure they are compatible, the results are now logged to the PC.

❑ **Zero Key Handling.** On upgrades and exports, Presto will convert IDs in PR_CUST with a value of 0 to NULL. This prevents Change Management issues where zeros are handled differently than NULLs.

Additional Notes:

- On upgrade, Server API will default to **ON**. The IDE will display a confirmation dialog on attempts to uncheck Use API checkbox in the hidden settings.

Presto 9.1 New Features

❑ **Navigable Logs.** Presto has added the ability to navigate backwards through a log, using the Page Up button.

❑ **Signed Files.** The Installation .exe files are now signed, which should prevent triggering of anti-virus programs.

❑ **IDE Enhancements:** The following features have been added:

- The IBM i installer now has a link to the Knowledge Base, to guide you through the FTP errors during the install.
- The Spool File Viewer has updated icons.

Presto 9.0 New Features

❑ **Cloud License Key Support.** Presto license keys now allow wild cards for LPAR specific values. This supports environments, such as Cloud providers, where these values may change.

❑ **Global Transformations enhancements.** The following features have been added:

- Export/Import global transformations in JSON format.
- **REGEX support.** Ability to use regular expressions in the search criteria. The default suite now includes 4 new Global Transformations that use REGEX to detect dynamic text like dates and times.

- **Element Type Filter and Restrictions.** Ability to limit your search criteria to a specific element type. You can also apply your transformations to next or previous elements that are specific element types.
- **Insert Elements.** Create transformations and insert them into screens that match your search criteria.

Please see "Global Transformations" on page 265 of Chapter 2 for documentation of the Global Transformations.

❑ **New Features Splash Screen.** Every new version of the Designer will display a Splash Screen upon initial startup, highlighting the latest new features in Presto. This screen can also be found under the menu option Help → What's new?

❑ **STRPCCMD Enhancements.** The following features have been added:

- New custom handler to launch web links without using the STRPCCMD Windows app. This is useful for customers with a security policy that does not allow windows applications.
- Improved whitelist error message handling.

❑ **Improved Window/Table Detection.**

- New WindowFinder version that supports horizontal window borders made from '-' characters as well as '+' for border corners..
- Ability to disable caching of detected table details. It is useful when, due to panel detection settings, different screens are matched as the same screen and one of them has a table on it, which then will be displayed on all the screens.

❑ **SmartChart Enhancements.**

- FusionCharts have been updated to v3.16, and introduces 7 new "Theme" options. Also included is a new "Pareto" Single Series Chart type.
- As a result of the update, a bug has been fixed where the font-family properties were not being applied to the SmartCharts. As a result of this fix, existing charts will display the correct default fonts whereas before they would show a browser default font, and captions would not be displayed in bold. Further, the font family and caption font family chart element properties editable in the Visual Editor now work to change the displayed font.

❑ **Login Macros.** Presto now has the ability to automatically run a macro after logging in.

❑ **Improved SQL Error Handling.** Presto now displays an error message when a SQL fetch returns bad data, and outputs the record number where the fetch failed.

Presto 8.9 New Features

Highlights for this release are:

❑ **Table/Window Editor Green Screen View.** The Visual Editor's Table/Window editor now displays in a monospace font, to ensure correct alignment when using non mono fonts on custom windows and tables.

❑ **Enhancement to Presto.ExecuteSQL function.** Support for the "complete" function has been added. This allows you to change the cursor to a "waiting" or "loading" cursor to indicate while an SQL statement is running. The function will be triggered once the request is complete regardless of whether the request was successful. It will trigger after either the 'success' function or the 'error' function depending on success.

❑ **Global Transformations enhancements.** The following features have been added:

- **Added Name and Description fields.** This allows you to create custom labels on the global transformations, making them easier to identify.
- **Added Default Global Transformations to provide modern views.** This creates cleaner, minimalized screens in Presto out of the box. These are inactive by default.

Presto 8.8 New Features

Highlights for this release are:

❑ **IASP Support** The Presto Installer now gives you the option to specify an IASP and the path in which to install Presto. This gives customers the flexibility to install Presto environments on different IASPs, allowing numerous configurations.

❑ **Presto REST API Kerberos Support** New REST API sessions are now able to use Kerberos authentication to automatically login to a Presto session. Customers using Kerberos have the ability to utilize Presto REST API on a secured connection.

❑ **Spool File Viewer Enhancements.** The following features have been added:

- **Select OUTQ and User Search Options.** You can now use Queue and User search options in the Spoolfile Viewer. These new options allow you to change your spool file list to the desired Queue and/or User on the fly.
- **Support HTML bold text.** The Spoolfile Viewer now has the ability to display bold-faced text when spool files are viewed as HTML.
- **Support for AFPDS spool files.** You can now view AFPDS spool files in PDF format (note that view as Html is not supported). This requires a minimum OS of 7.1, with PTF SI43471. For details, see https://www.ibm.com/support/pages/using-cpysplf-re-create-spooled-file-or-create-pc-file in IBM Support.
- **IASP Support.** The Spool File Viewer will now display Spool Files from all IASPs in your system.

❑ **Skin Editor enhancement.** Added an additional option "Logo Display". This css property allows you to display the logo as an inline-block element, or remove the element altogether. The new Display property for logos allows finer control of the logo space in the header.

❑ **Improved Right To Left Language Support.** Presto now has better support for the display of data in the browser for Right To Left languages such as Arabic (CCSID 420 or 8612) and Hebrew (CCSID 424 or 62211).

❑ **Server API Conversion.** As of Presto 8.8, most of the communication between Presto's client and server components will occur via a REST API. This can still be disabled from the hidden developer menu.

❑ **Upgrade JQuery.** The version of JQuery used by Presto has been updated to v3.51. The benefits of this upgrade include better performance and support for the latest developments and plugins. The new version may throw exceptions in some cases where the old version would fail silently.

Please call customer support if you have any questions.

Presto 8.7 New Features

Highlights for this release are:

❑ **Presto REST API.** Allow more advanced customers to interact with a Hub session from start to end using a REST-like API.

❑ **IDE Improvements.** Various updates to the IDE for enhanced usability.

- **Skin Editor:** The option to change logo height and alignment (left, right, center) has been added to the skin editor.
- **Page Management:** Presto now has the ability to delete multiple pages at a time.
- **Identities Page:** The control panel in the Identities page is now resizable.
- **Upload:** Upon successful upload of files or images from the Tools menu, the notification dialog now displays the full IFS path and name of the file.
- **Settings UI:** The Global Settings tab has been redesigned to fit all options on screen.
- **Server Logs:** Presto now supports the use of table data in debug logs. This enhances table detection when replaying logs within the IDE, and is useful for debugging.

❑ **Check the Device Name.** The device name can now be retrieved from any page with the Presto.GetDeviceName() function. This is useful for determining whether the session is being run in Presto or an emulator.

❑ **SQL Register.** Presto now sets the client_userid register in the SQL connection to the signed in user. This is useful for supporting options that depend on the current user, such as the choices displayed in a drop down list based on who is logged into the session.

Presto 8.6 New Features

Here is a summary of the most important features in Presto 8.6. As is the case with each release, these changes are supplemented by dozens of smaller UI and performance improvements.

❑ **Convert Spool File to HTML.** Presto now has the ability to display spool files in HTML format. This option is available when a user clicks on an entry in the spool file viewer. You can select "View in PDF" or "View in HTML".

❑ **Global Transformations - On Transform Event.** New global transformation event "onetransform". This event extends global transformations functionality by adding the ability to

execute custom JS logic following a global transformation. This is useful when the user wants to apply exceptions to global transformations.

❑ **JavaScript Events - SentRequest and PlayMacro.** Presto has 2 new JavaScript events, SentRequest and PlayMacro.

- **SentRequest** is triggered when a page request is in progress, after a SendRequest event. SendRequest is an event notifying the user that the request is about to be sent and giving them the opportunity to modify or cancel that request. SentRequest is an event notifying the user that the request has been sent to the server. Unlike SendRequest, this event cannot be cancelled as the request has already been completed.
- **PlayMacro** is a new event triggered before a macro is played. These events are useful for triggering actions after a request to the server has been sent. For example, executing custom code on page navigations.

❑ **Check For Attached Customizations.** Determine whether a customization has been attached to a screen with the new Presto GetScreenProperty() function. Paired with SetScreenProperty(), these functions replace the now deprecated GetEnvironmentSettings() and SetEnvironmentSettings(), and introduce new options of HasTransorm, HasCustomHtml, and HasJavascript. The complete suite of options is useful for getting and setting variables related to the current screen.

❑ **Change Management Export - Exclude Configuration Settings.** To support third-party methods to synchronize customization files between environments, the option to export customizations without the contents was added in Presto 8.3 . In addition, this feature will now exclude environment configuration settings from the export files (js settings, config.bws settings, etc). During the export process, if the "Include contents of customization files" box is unchecked, environment configuration settings will not be included in the .xml file.

❑ **Enable Browser Right-Click Menu.** The option to enable the browser's right click menu has been added to Presto. This allows users to access features in Chrome and Internet Explorer such as "Translate" and "Inspect".

Presto 8.5 New Features

Here are the most important features in Presto 8.5:

❑ **Flyout Transformation.** Presto now has the ability to transform an input to a flyout menu. The flyout appears when you hover over an input.

❑ **New SQL Queries Interface.** There is a new SQL Query tab interface to work with your SQL queries. To open this interface, Select Tools>SQL Queries.

❑ **Support finding all instances for Global Transformations.** Global Transformations (Tools>Global Transformations) has new options for determining what elements match the screen.

❑ **StrPcCmd - Regular Expressions.** In the Expressions tab in the StrPcCmd, you can now specify regular expressions that it will match the submitted command. If the command matches the regex, it will run without asking you to confirm.

❑ **StrPcCmd - Import and Export Settings.** The StrPcCmd has the ability to import and export settings found in the StrPcCmd utility for clients who need to set it up on multiple PCs.

❑ **Create Page Up and Page Down buttons when no Subfile is present.** Presto now has the option to create page up and page down buttons when there is no subfile present or subfile detection is disabled.

❑ **REST API Support.** This version includes support for some of the communication between the client and server components to occur via a REST API.

❑ **JMeter Support.** The IDE install includes a JMeter test script to help test performance in Presto. See Knowledge Base article 16430 for instructions on how to get started using this script.

Presto 8.4 New Features

Here are the most important features in Presto 8.4:

❑ **Global Transformations.** Global Transformations are a new feature that allows transformations to apply site-wide based on the text of a given screen. Text reoccurring throughout your site can be found and applied a transformation.

❑ **Export Screens Changed After a Specified Date.** The command-line program PR_CHGMNG now supports a new parameter changedsince that allows you to export screens changed that have only been changed after a certain date.

❑ **Code Snippet Panel Added.** The JavaScript Code Snippet view is now also accessible from the Visual Editor Event Editor and Custom HTML pages.

❑ **Bypassing "Display Program Messages" Screen.** This is a new config.bws option (Tools>Advanced Configuration) that allows you to bypass the "Display Program Messages" screen when a user signs on to multiple sessions.

❑ **Table Editor - Manual Entry of Setttings.** When editing a table, you can now enter values for the start row, start column, and table height in the element properties panel.

❑ **Window Editor - Manual Entry of Settings.** When editing a window, you can now enter values for the start row, start column, width, and height in the element properties panel.

❑ **Option to Download All Insight Files.** Under Help>Insights Recordings, you can now download all insight files at once to your PC.

❑ **Filtering the Transformation List.** We've added a filter box to the transformation list in the Visual Editor so that it's easier to find a transformation applied to your page.

Please call technical support if you have any questions.

Section 2 - Introduction and Installation

CHAPTER 4 - INTRODUCTION TO PRESTO 2
Benefits of Using Presto 2
How Presto Works 4
Key Features 5
Get More Out of Presto with other Fresche Solutions 6
CHAPTER 5 - INSTALLATION AND CONFIGURATION 7
Installation Overview 7
Starting the XL_PRESTO Apache HTTP Server Instance After an IPL 11
Starting the Exodus55 Subsystem After an IPL 12
Starting the License Server Subsystem After an IPL 13
Datepicker Localization 20
Moving On 23

Chapter 4 - Introduction to Presto

Using Presto to give all your green screens a modern Web GUI is the fastest way to keep your IBM i relevant in the eyes of your end users, partners and upper management. They will love the visually appealing screens with their new functionality that are easier to use and accessible from popular browsers, tablets and handheld devices. With a more modern and familiar interface to your applications, training new users also becomes much simpler.

Presto includes a Windows-based design tool you can use to add new functionality, such as images, datepickers, drop down lists and tabs to your web-enabled screens. You can do all of this without coding HTML, which reduces the web development learning curve for RPG or COBOL programmers. Your changes appear on the fly so you can review your results instantly while comparing your changes with the original, untouched green screen versions.

Benefits of Using Presto

Here is a quick overview of some of the benefits of using Presto to modernize your applications.

Immediate Web GUI

Presto gives your existing green screens a web GUI much faster than rewriting your programs. As soon as you complete the server-side installation of the product (which typically takes about ten minutes) all of your existing green screens, including RPG and COBOL programs, menus, windows, third party and system screens, are accessible as modern looking web pages. With Presto, you don't have to go through any complicated screen, database or application mapping and identification processes to get running.

Function keys and menus are automatically transformed to buttons and links, subfile options become clickable flyout menus, and you can access spool files from your screens with the option to view them as PDFs.

You can even choose one of several modern skins that you can customize to match your company's look. Because Presto does so much out of the box, you will have more time to add new functionality to the screens that you want to further enhance. Many Presto users take a phased approach to their modernization projects. For the majority of their screens, they choose to deploy the consistent look and experience and instant results that Presto's 5250 datastream provides on installation. They may also give their new RPG programs a web GUI using RPG OA.Then they further enhance and add new functionality to their most popular screens.

New Functionality

You can increase the life and effectiveness of your applications by adding new UI elements, such as auto-completes for faster data entry and charts for visualizing data, to your web-enabled screens.

Presto's Windows-based design tool, the Presto Designer, includes a Visual Editor and makes it easy to customize both individual screens and your environment as a whole. Using the Visual Editor you can move elements anywhere on the screen and add new functionality like links, images, datepickers, drop down lists, graphs and much more. You can also extend your screens beyond the 5250 datastream by accessing additional data with AJAX and SQL.

While Presto includes useful tools for editing your web pages, the software also allows you to work directly with the HTML, CSS and JavaScript within the Presto Designer. Programmers with web development experience will enjoy Presto's flexibility and the control they have over the code delivered to the browser.

Support for RPG OA

Presto supports RPG Open Access (OA), automatically building jQuery grids for your subfile data. See "Using Presto with RPG OA" on page 335

Existing Programmers and IT Staff Can Use Presto

Programmers and IT staff with no web development experience are productive right away using the Presto Designer to further enhance screens. The Presto Designer will introduce you and your staff to the exciting world of web application design and publishing, using your robust existing applications as the foundation.

Presto can also serve as a transitional stage for your organization by exposing you to many of the concepts and web technologies you and your staff will need to be familiar with if you decide to head in the direction of standard web programming using PHP, RPG and tools like our award-winning WebSmart.

No Code Changes

With Presto, all of your green screens, including your RPG and COBOL programs, menus, third party and system screens, are instantly accessible as web pages. You don't need to make any changes to any RPG, COBOL or DDS source. You don't even need the source! Presto also works with all versions of RPG (II-IV).

Presto is truly non-invasive in that you can continue running your existing 5250 programs unchanged. They are not in any way affected by Presto. All of your proven code is left untouched, reducing the risk and time needed to modernize. You can also configure Presto such that it eliminates any 5250 OLTP costs and only uses your batch processor.

No Additional Software Required

Your IT staff saves time because users only need a browser to access the web-enabled screens internally or remotely from PCs, Macs or mobile devices. No ActiveX, Windows Servers, WebSphere or other software are required.

The Presto Designer includes all the tools and flexibility you'll need to add new functionality to your screens, ranging from the easy-to-use Visual Editor to full control over the HTML, CSS, JavaScript and jQuery. You don't need to know PHP, have web development experience, or use any additional web development tools to further enhance your screens.

Reduces the Cost of Training New Users

It can take a long time to train users who are new to the IBM i and its unique navigation characteristics like subfile options and function keys. New users will often prefer modern web pages over archaic green screen applications. They will be more comfortable with the modern web pages produced by Presto, which are easy to access and have images, mouse-driven navigation and intuitive input fields like drop down lists.

Enterprise-Ready Solution

Critical to any enterprise solution, Presto supports multiple environments and languages. Presto environments are a set of global settings for a group of programs or screens on your system, providing you with test and production environment(s).

If you are an Independent Software Vendor (ISV) who needs to web enable your existing 5250 solutions, Presto environments give you the ability to simultaneously have independent versions of the same web-enabled apps. ISVs can use environments to make the same application have a customized look for each client.

How Presto Works

With Presto's 5250 datastream appear, you can access all your applications out of the box in any PC or mobile browser without additional components, including the source code. You don't even need to modify the source code in any way. From this starting point you can then add custom enhancements to your screens using the Presto Designer or by directly modifying the HTML, CSS and JavaScript.

You can also use Presto's RPG OA handler to give new RPG programs a web GUI. Please see "Using Presto with RPG OA" on page 335 for more information on how Presto's RPG OA handler works.

Presto Run-Time Architecture

Here is an illustration of how Presto works:

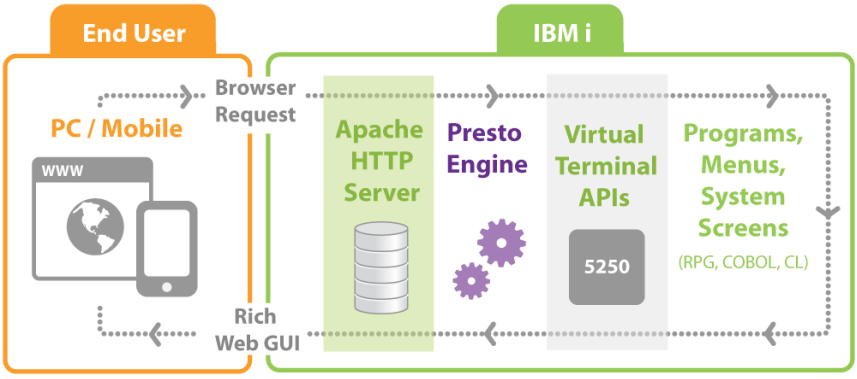


Figure 2-1. Presto architecture diagram.

Presto works like this:

1. The user navigates to a URL in their browser on their PC or mobile device. That URL points to the IBM i.
2. The browser communicates with the IBM Apache HTTP Server which processes the request and evokes the Presto Engine.

3. The Presto Engine starts the Virtual Terminal session. It also processes requests in both directions, rendering your green screens as web pages to the browser by transforming the 5250 data stream into HTML, CSS and JavaScript and returning data in the format that the Virtual Terminal requires.
4. The original RPG or COBOL programs and menus run in the background. Because Presto works on the 5250 data stream level, your programs are not affected in any way and can still be accessed on 5250 emulators by users who prefer green screens.

Key Features

Other Presto key features include:

Presto Designer and Visual Editor

You can improve the look and functionality of screens even if you have no web development experience.

- Add datepickers, images, tabs, drop down lists and much more.
- Drag & drop or remove text and fields.
- Add links to PDF files, Word documents and other files.
- Format text and field properties.
- Add JavaScript events for custom user interactions.

Global Settings

You can apply settings or create custom rules to enhance all of your screens without touching them.

- Customizable skins (colors, images and styling).
- Clickable menus and function keys.
- Formatted subfiles, date fields and program titles.
- Ability to create test, production and other environments.
- Support for Non-English IBM i language settings.

Database Driven UI Elements

You can extend screens beyond the 5250 datastream using SQL to access data that's not in the original program.

- Drop down lists, auto completes and charts.
- Dynamic SQL using values from the screen.
- Export data as CSV spreadsheets or HTML.
- Add new data to display or update.

Mobile Support

You can access your web-enabled applications from iPads, iPhones and other tablets and smartphones.

- Automatically detects the type of mobile device
- Conditionally uses mobile-friendly layouts.
- Includes a virtual keyboard for function keys.
- Flexibility to add touch gestures.

- Ability to reconnect to a page if a signal is temporarily lost.

Responsive Support

You can build responsive screens that resize and show or hide content when you access them from different devices - including iPads, iPhones and other tablets and smartphones.

- Includes a Responsive visual editor that can be used to design pages that change layout depending on the size of the browser window.
- Optimally display content for different platforms such as iPads, iPhone, Android tables, desktops and more.
- Customizable skin (colors, images, and styling) that provides a modern responsive look and feel.
- Uses Bootstrap , an HTML5-based framework for building responsive websites.

HTML, CSS and JavaScript

You have full access to client-side code so you can create richer web screens with these modern technologies.

- JavaScript libraries like jQuery.
- JSON, AJAX and XML.
- HTML5 and CSS3.
- External plugins.

Web Application Integration

Presto can communicate with other web apps (PHP, RPG, etc.) for a more sophisticated and seamless experience.

- Access web services for Google Maps, credit card approval, tracking shipments *and more*.
- Navigate and pass data to other web applications including WebSmart programs, Clover queries, or any other web app (PHP, .NET, etc.)
- Provide access from portals, intranets and extranets.
- CSRF (Cross-Site Request Forgery) protection and warnings.

Get More Out of Presto with other Fresche Solutions

Presto is available as a standalone solution or as part of Fresche's modernization suite.

Nexus Portal

Presto optionally includes Fresche's Nexus Portal. Nexus provides a secure single point of access to your enterprise information including your Presto web-enabled apps, other web apps and pages, documents, dashboards and productivity tools. Nexus includes a web menu system and single sign-on for your Presto apps that you can use to hide traditional 5250 navigation.

The next chapter will help you get Presto installed as quickly as possible.

CHAPTER 5

Chapter 5 - Installation and Configuration

Follow the instructions below to install Presto 10.0. If you are upgrading from an earlier release, use the instructions in the New Features & Upgrade Guide.

Installation Overview

Presto includes several components:

1. On the IBM i these libraries are installed:
   - **XL_PRESTO.** The default Presto product library name. This name is also used for the IFS folder and HTTP server instance.
   - **XL_PRESTOI. The Presto installation library.**
   - **XL_PRDEMO.** The Presto demo programs library.
   - **XL_EX55.** The Client/Server Communications Library.
   - **XL_LCSRV.** This is the License Key Server Library.
   - **XL_WEBSPT.** Run-time objects used by Presto.
2. An IBM i IFS folder [esdi/xl_presto].
3. An Apache HTTP server instance [also XL_PRESTO].
4. On your PC you will install the Presto Designer.

You can remove the library XL_PRESTOI from your system at any time after installing the software. You only need XL_PRDEMO to work with the tutorial.

Your Presto License Key

You need a Presto trial period license key to use the software. If you don't have a license key phone Fresche at 250-655-1766 or email support@freschesolutions.com to request one. You can enter the license key during the install, or afterward by adding library XL_PRESTO to your list and running the UNLOCK command:

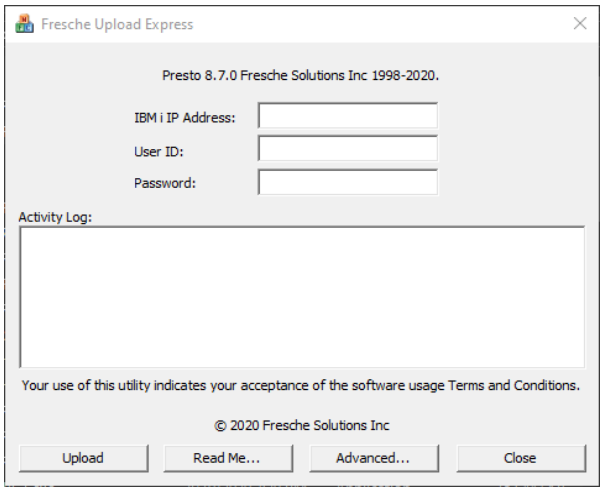
```
ADDLIBLE XL_PRESTO

UNLOCK
```

Step 1: Download Presto and Transfer the IBM i Install Files

Follow these steps to start downloading and installing Presto:

1. Your Fresche sales rep will send you an email with instructions for downloading the most current version. Download both the server and PC installation files and save them to your PC.
2. When the downloads finish double-click the IBM i install executable file named prestoserver-10.0.exe to unpack the files and begin transferring them to your IBM i. [The exact file name may vary.]
3. When the window below appears, enter your IBM i IP address or domain name, and user profile/password:



4. The upload utility presented when you execute the IBM i install file.
Click the Upload button. When the upload finishes the activity log displays the message 'File upload complete'. You can then close the Fresche Upload Express window shown above.



If the window above doesn't appear automatically, execute the file install.exe from the folder to which you unpacked the Presto download. If you get an FTP transfer error, please visit our Knowledge base and search for the error you encountered. You may need to adjust the FTP settings on your IBM i.

Step 2: Install the Presto IBM i Components

If this is the first time you have run a Presto install on your system, you will need a SECOFR profile with *ALLOBJ, *SECADM, and *JOBCTL special authorities to run the install. Follow the instructions below to install the Presto IBM i components.

1. Log on to an IBM i session and execute the command `XL_PRESTOI/BCDINSTALL`.

2. At the first screen presented select option 1 for a new install:

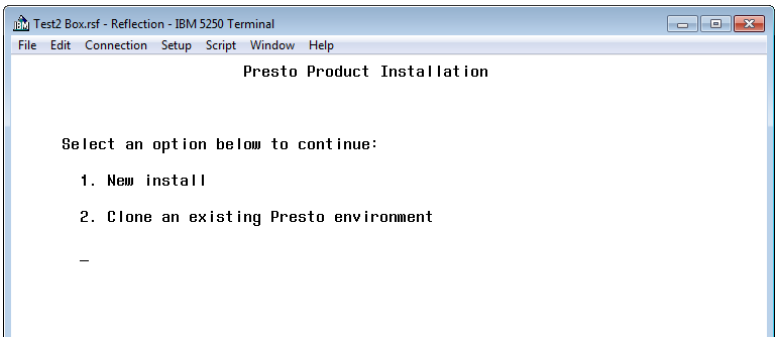


Figure 2-1. Choosing a new installation or an upgrade.

Press Enter to continue.

3. The install will prompt you for an environment name to create. The default is `XL_PRESTO`. If this is your second, or subsequent, install, enter a different name (e.g. `XL_PR100`, or another unique name).
Press Enter to continue.
4. When prompted, confirm the IFS directory to be created for your Presto environment. The default directory is /esdi/xl_presto (or the environment name you chose).

   Then press Enter to continue the installation.
5. When prompted, confirm the Apache HTTP server configuration to be created in your IFS. The default location is /www/xl_presto (or the environment name you chose).

   Press Enter to continue.
6. When prompted, enter the license key provided to you.



If you are installing Presto for the first time you can enter the license key during the install or afterwards using the `UNLOCK` command in your Presto environment library (`XL_PRESTO`, unless you gave it another name):

```
ADDLIBLE XL_PRESTO
UNLOCK
```

Press Enter to continue restoring the remaining objects.

When the IBM i installation finishes the Installation Status screen presents a summary of the library, IFS directory and HTTP server instance created, similar to this one (the release levels and library names may vary):

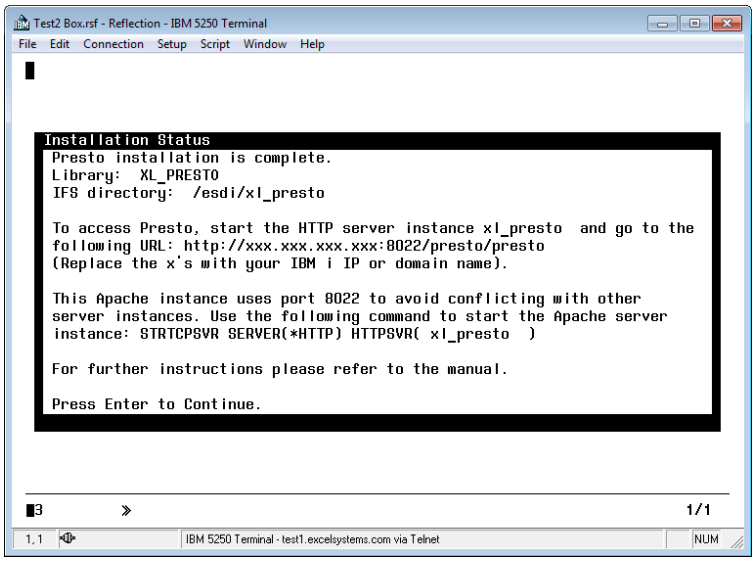


Figure 2-2. Confirming the installation settings.

If you like you can print this screen or make a note of the values.

Then continue with the next section.

Step 2b: Install the Presto IBM i Components on iASP (optional)

1. Log on to an IBM i session and execute the command `XL_PRESTOI/BCDINSTALL`.
2. The installer has a new field in the Install screen. Use this field to specify the name of the ASP you are installing to, for example IASP1 (default is *SYSBAS).
3. The next screen confirms the IFS directory to be created for your Presto environment. When installed on an ASP, the default directory is `/iasp1/esdi/xl_presto` (or the iASP and environment names you chose).
4. There is a new screen that prompts you to enter the IFS for the Apache config. The default when installing to an ASP is `/iasp1/www/xl_presto` (or the iASP and environment names you chose)

IASP Considerations

1. The installer will check that the selected IASP both exists and is accessible to the user before proceeding.
2. The Exodus Job Description will need to be updated for the correct ASP group.

   If you install a Presto library to an IASP, the Exodus user will need to be configured to access this IASP via the ASP group name. To update the Exodus Job Description call `WRKJOBD XL_EX55/EXODUS`, take a 2, page down to Initial ASP Group and set it to the correct ASP group for the IASP.

   Only one ASP group can be set at once so if you have Presto Environments (the libraries specifically) on IASPs in different ASP groups, the Exodus Job Description would need to be changed whenever a user wanted to develop in a different environment.

3. How to find the Apache config if you forgot where you installed it.

The Presto installer allows you to change the location where the Apache config will be installed, so that you can put the config folder on the IASP. If you had changed the location and then forget where it is, you will need to check the Apache registry for it.

To access the registry, use STRPDM, take a 3, enter the file name QATMHINSTC and library QUSRSYS. Then find the member corresponding to the Presto environment. The member name should match the Presto library name.

Take a 5 on that member.

4. Cloning from one ASP group to another is currently not possible.

   If the user installs a library to an IASP in one ASP group, then attempts to clone that library and place the new library on an IASP in a different ASP group, this will not be allowed by the installer.

   The reason is that the job can only have one ASP group set at a time so if 2 libraries are on IASPs in 2 separate ASP groups the job can only see one at a time.

Step 3: Start the Presto HTTP Server

1. The IBM i installation creates an Apache HTTP server instance with the same name you gave to your environment library in Step Step 2 (the default is XL_PRESTO).
2. All new Apache server instances are set to run on port 8022 by default. If you are installing multiple versions of Presto you need to make sure each HTTP server instance is referencing a unique port number before starting it.



This port number is the one used by the Apache HTTP server to display your pages in a browser. This is NOT related to the port number used by the Presto Designer to connect with the IBM i server components.

To start your Presto server instance (XL_PRESTO, for example) run this command:

```
STRTCPSVR SERVER(*HTTP) HTTPSVR(XL_PRESTO)
```

After a few seconds, you can confirm that the HTTP server was started by running this command:

```
WRKSBSJOB SBS(QHTTPSVR)
```

You should see several jobs named XL_PRESTO (or your selected environment name) in the QHTTPSVR subsystem.

See the section “Working With the Presto Apache HTTP Server Instance” on page 401 for further details about the HTTP server configuration.



Starting the XL_PRESTO Apache HTTP Server Instance After an IPL

If you IPL your IBM i nightly or on the weekends you need to also restart your XL_PRESTO Apache HTTP server instance afterward.

During your evaluation period you may elect to do this manually each time you work with the tool. However, if you purchase the software or use it for a lengthy evaluation period you should add this step to your startup procedures. You can find further information on this in our online Knowledge Base, in article 8816:

https://www.myfreschesolutions.com/kb?kbnum=8816

Step 4: Start the Exodus55 Subsystem

Presto uses a subsystem named EXODUS55 to transfer data between the PC design tool and the IBM i. The subsystem contains an auto-start job that runs constantly to monitor and dispatch conversations to the appropriate server programs.

When the server side installation finishes use the WRKACTJOB command to confirm the EXODUS55 subsystem has been started.

If this is a second install or you already have another Fresche tool installed on your system that uses EXODUS55, the install will not start the subsystem automatically. In these cases, if the subsystem is not already running, use this command to start the subsystem now:

```
STRSBS XL_EX55/EXODUS55
```

Once the subsystem is running continue to the next step.



Starting the Exodus55 Subsystem After an IPL

If you IPL your IBM i nightly or on the weekends, you need to also restart TCP/IP services and the EXODUS55 subsystem. While you probably already have processes in place to restart TCP/IP, you can add the command STRSBS XL_EX55/EXODUS55 as part of that sequence.

While you are evaluating the software you may elect to start it manually. But if Presto isn't working Monday morning, this is probably the reason! Find further information on this in our online Knowledge Base, in article 8816:

https://www.myfreschesolutions.com/kb?kbnum=8816

Step 5: Start the License Server Subsystem

Note: Starting at version 8.4, the License Seat Count will be enforced.

Although the license server should have started automatically during the install process, it makes sense to double check that. Use the WRKACTJOB command to check if the LICSRV subsystem has been started and that it has LICSRV job running under it.

In case the subsystem is not running, you can start the subsystem by using this command:

```
STRSBS XL_LCSRV/LICSRV
```

Once the subsystem has been started continue to the next step.

Starting the License Server Subsystem After an IPL

If you have routine IPLs scheduled for your IBM i, you will need to start the License Server after each IPL. You should consider adding the command `STRSBS XL_LCSRV/LICSRV` to your start-up CL program.

For more information about the License Server, please refer to this Knowledge Base article:

https://www.myfreschesolutions.com/kb?kbnum=16087

Step 6: Confirm Your Server Installation

To confirm your server side installation and begin using Presto open an Internet browser session with any modern browser and enter this URL:

```
http://xxx.xxx.xxx.xxx:8022/presto/presto
```

Replace the `xxx.xxx.xxx.xxx` with your own IBM i domain or IP address (for instance 'intra.company.com' or '192.168.0.100').

Port 8022 is the default value for your first installation; for subsequent environments make sure you change the corresponding Apache HTTP server instances to use a unique port.

Here is your IBM i Sign On screen using Presto's default settings:

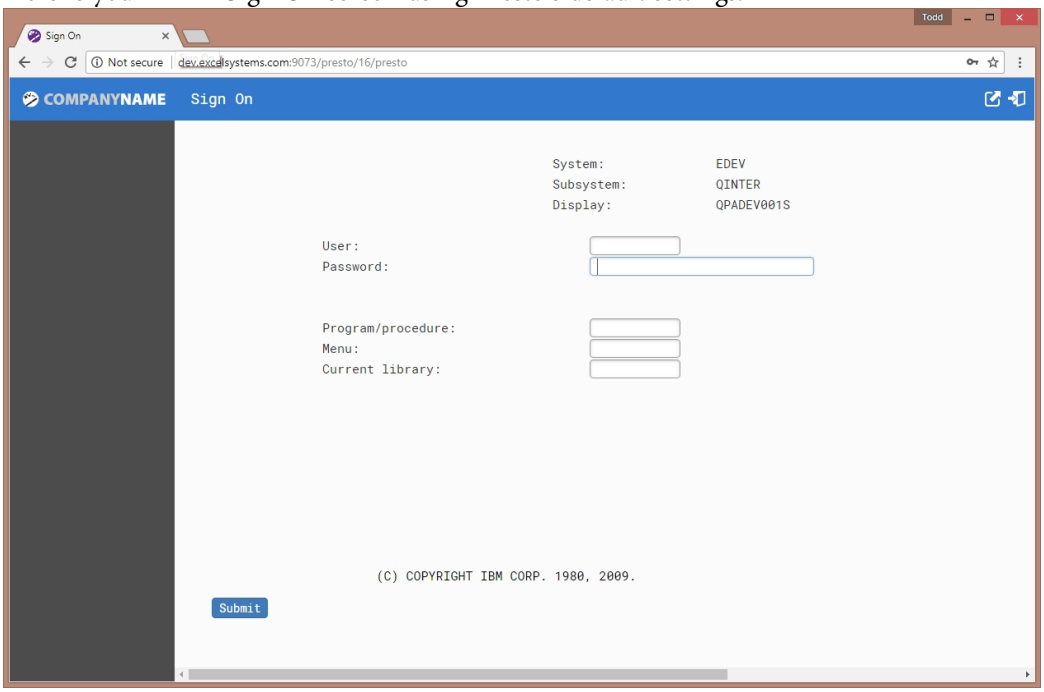


Figure 2-3. An IBM i Sign On screen, presented using the Presto default configuration.

Note that in the following pages we use illustrations from a variety of Presto skins and environments on different servers, so your own screens may vary slightly from those illustrated here.

If you don't get a successful connection you can try the troubleshooting instructions below or complete the PC install and then call Technical Support for help.



Here is information about a few system configuration settings that you can verify if you don't initially get a successful connection.

QTMHHTTP's MSGQ

In order for you to use Presto the user profile `QTMHHTP1` must have `*ALL` authority to `QTMHHTTP`'s message queue (`MSGQ`).

To check this run the command `WRKOBJ OBJ(QTMHHTTP) OBJTYPE(*MSGQ)` and select an option 5 to display authority. If `QTMHHTP1` is not in the User list or has an object authority other than `*ALL`, run this command to edit `QTMHHTTP`'s `MSGQ` authority: `EDTOBJAUT OBJ (QUSRSYS/QTMHHTTP) OBJTYPE(*MSGQ)`.

QUSRWRK Subsystem

Some Presto jobs must run within the `QUSRWRK` subsystem, which must be active. Run the command `WRKACTJOB SBS(QUSRWRK)` to check this. If it isn't running, start it with the command `STRSBS QSYS/QUSRWRK`.

Virtual Terminals

This is not all that likely to arise, but to use Presto you must have enough available virtual terminal jobs. Check this using the command `DSPSYSVAL SYSVAL(QAUTOVRT)`. The value shown should represent at least 3-5 times the maximum number of users you anticipate could be connected to your system. If it is 0 or seems low, prompt the command `CHGSYSVAL SYSVAL(QAUTOVRT)` to change it to a higher value.

Step 7: Install the Presto Designer

Follow the steps below to install the Presto PC design tool.

1. Download the latest version of the Presto client to your PC. When the download finishes execute the downloaded file (PrestoClient-10.0.exe) to unpack the install files and initiate the install.



Presto 10.0 requires that the Microsoft .NET Framework 4.0 be installed on your PC.

Because Microsoft considers the .NET Framework 4.0 to be an optional update it's possible that you won't yet have it on your PC. If that is the case you need to install it before you can complete the Presto 10.0 install.

This is a large file that may take some time to download and install. For more details search http://www.microsoft.com for ".NET Framework 4.0".

Subsequent Presto PC installs should only take a few seconds.

---

2. This window appears:

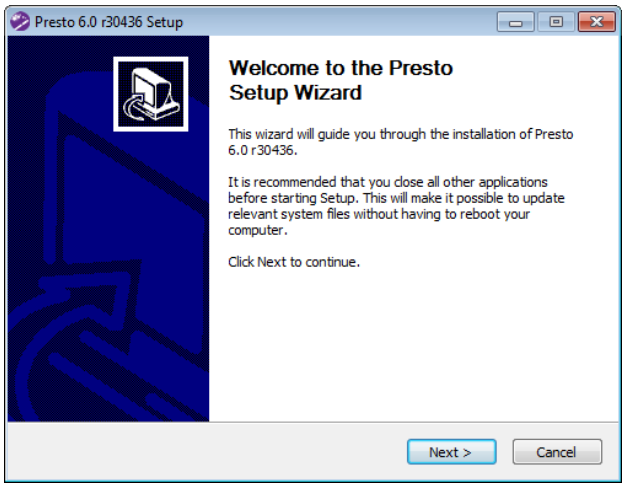


Figure 2-4. Installing the Presto Designer on your PC.

3. Follow the instructions to install the Presto executable to your PC. When the installation is complete, click Finish.

The install will give you the option to launch the software immediately. You can also launch the Presto Designer from Start>Programs>Presto>Presto.

Continue with the next step to configure the connection to the IBM i.

Step 8: Configure the Presto Designer to Connect to the IBM i

To work with your applications using the Presto Designer you must create a connection to the IBM i so the design tool can communicate with the server. Follow these steps to set up your server connection.

1. On startup the Presto Designer checks whether you have configured a server connection. If not, it prompts you to add one:

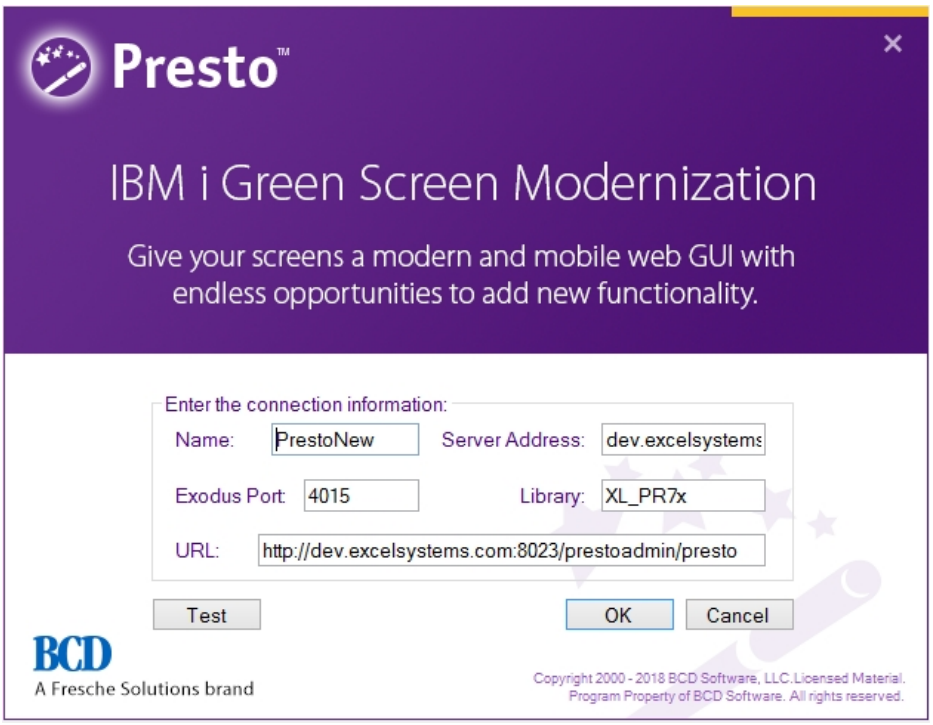


Figure 2-5. Configuring a Presto connection.

Complete the values as described below:

Field	Value
Name	Enter a descriptive value for this connection.
Server Address	Type your IBM i IP address or domain.
Exodus Port	The default value of 4015 should usually work for you. See the note below for details.
Library	The default install library is `XL_PRESTO`. Use this value unless you specified something different.
URL	http://yoursite:8022/prestoadmin/presto is the default value for this field, but the correct URL should be defaulted in after you move away from the Server Address field.



Port 4015. The default `EXODUS55` port of 4015 usually works. If port 4015 is in use you can change the default port for incoming connections on the server using the `XL_EX55/ZCFGTCP` command (press F4 to prompt the command, then press Enter). Then enter a matching value here.

If you change the server-side port value(s), restart the `EXODUS55` subsystem.

As noted earlier, this NOT the same as the Apache HTTP server port; the two are completely unrelated. This one is purely for the design tool.

2. After entering your settings use the Test button to verify your configuration. Presto will present the prompt below for you to sign on to your IBM i:

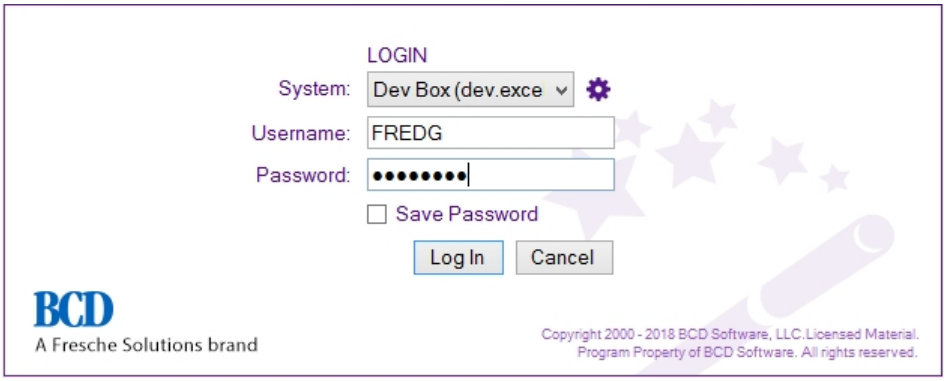


Figure 2-6. Sign on to your IBM i with the Presto Designer.

3. When you click OK Presto should confirm that you have connected.

Here are a couple of things to check if you don't get a successful connection:

- **Exodus55 subsystem is not active.** The most common connection error (10061) is caused by the `EXODUS55` subsystem not being active. If you get this error run the `WRKACTJOB` command and check whether the `EXODUS55` subsystem is running and contains a job called `EXSERV`. If the subsystem isn't started, run the command `STRSBS XL_EX55/EXODUS55`. If the subsystem is running but `EXSERV51` is not, end and restart the subsystem and confirm the job is added. Then try connecting again.
- **Other connection errors.** If the subsystem is up but you can't connect, add `XL_EX55` to the library list of your IBM i session and run the command `ZDSPERRLOG` to display a list of your connection attempts. Check for error messages (such as invalid User IDs, etc.).

If you can't spot the problem immediately please call Technical Support at 250-655-1766. You can also email us at support@freschesolutions.com.

---

4. After Presto verifies that your connection is working, click OK to save the connection. You'll then see it listed here:

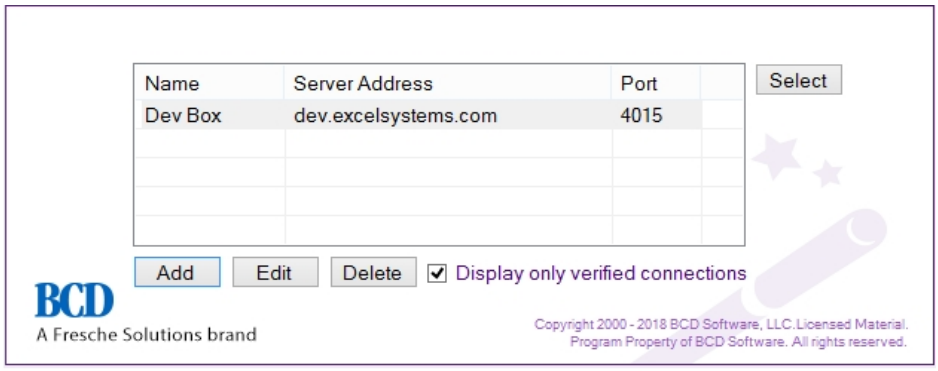


Figure 2-7. The Presto environment (server connection) you created.

5. Click OK again at the list shown above. When you do this Presto finishes launching the Designer.

After you log in the Presto Designer launches, displaying a web version of your IBM i login screen, similar to this:

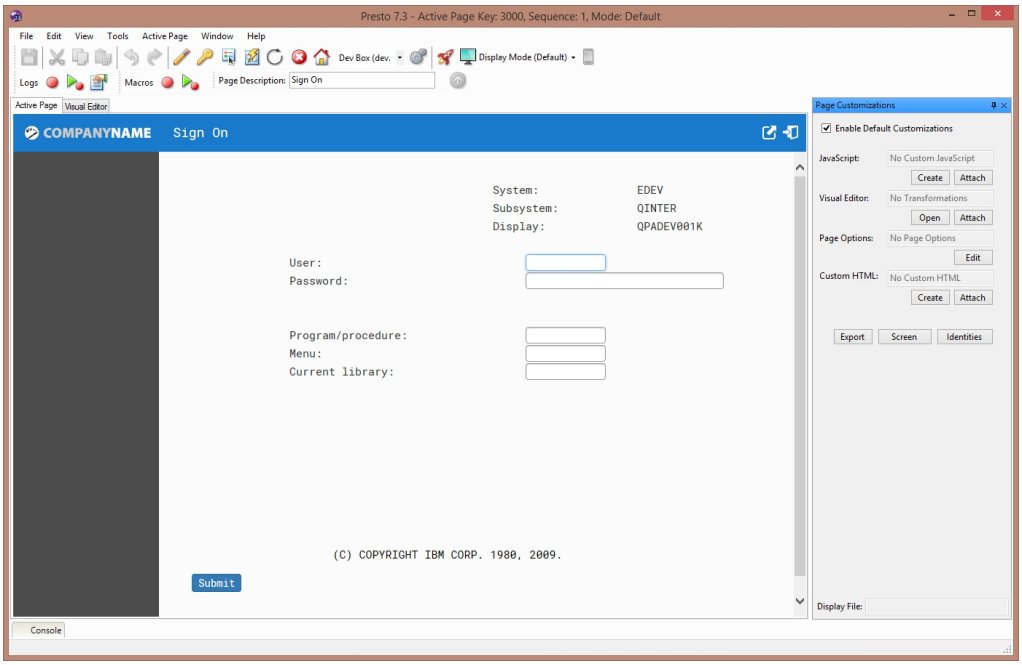


Figure 2-8. The Presto Designer with an IBM i Sign On screen.

Continue with the next step below.

Step 9: Review Your Server Settings

Every environment you create has its own server and detection settings. Select Presto's Tools>Settings menu option to work with these:

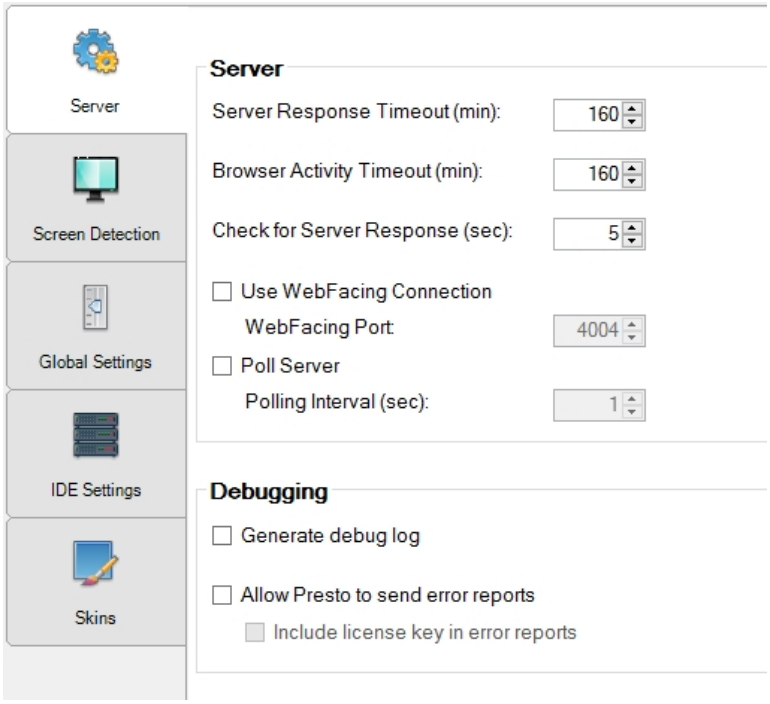


Figure 2-9. Reviewing your server connection.

The default values shown here and at the other tabs should work well enough for you to get started with Presto.

Your detection settings are critical to how Presto will work for you. Before you get too far into customizing your site read "Auto Detection Settings and Identities" on page 114 to make sure you understand how Presto uses these important settings.

You can find reference information for each of the options above in the section "Server Settings" on page 38.

For further information on using WebFacing see the section "Using WebFacing with Presto" on page 397. When you first install the software for your evaluation, we recommend that you NOT use the WebFacing as it adds dependencies on third party objects which can create some unnecessary configuration challenges at the outset.

Step 10: International Settings

There are a number of Presto configuration options for installing the software on a non-English language system or to support multiple languages.

This step is only required if your system has a CCSID other than 37!

Set Your Presto CCSID [*If Not 37]

Check your CCSID by running the IBM i command `DSPSYSVAL SYSVAL(QCHRID)`. If the value displayed for the Code Page is 37 you can skip to the next section.

If your system CCSID is something other than 37 you need to make a change to your Presto Apache HTTP server instance. You server instance name will match the environment name you specified in Step Step 2 (the default is `XL_PRESTO`).



If you aren't familiar with editing an Apache HTTP server config, please see the section "Working With the Presto Apache HTTP Server Instance" on page 401 for general instructions.

In your configuration find this line:

```
DefaultFsCCSID 00037
```

Change it to:

```
DefaultFsCCSID 00xxx
```

where xxx is your system's CCSID. Save your changes and restart the instance.

Virtual Terminals and CCSID Settings

With certain languages (e.g. DBCS languages) it is necessary that the virtual terminal session created have specific device description characteristics. If you need to configure Presto for one of these languages, please refer to the section "Virtual Terminals and CCSID Settings" on page 398.

Adding Support for Multiple Languages

To configure Presto to support multiple languages, refer to the instructions in the section "Configuring Presto for Multi-Language Support" on page 399.

Datepicker Localization

The datepicker you can implement via the Presto visual editor is set to display English language settings. If you are building your application for another country you can take advantage of a jQuery data localization feature.

To do this start by opening your custom.js file (Tools>User Resources, from the Presto Designer):

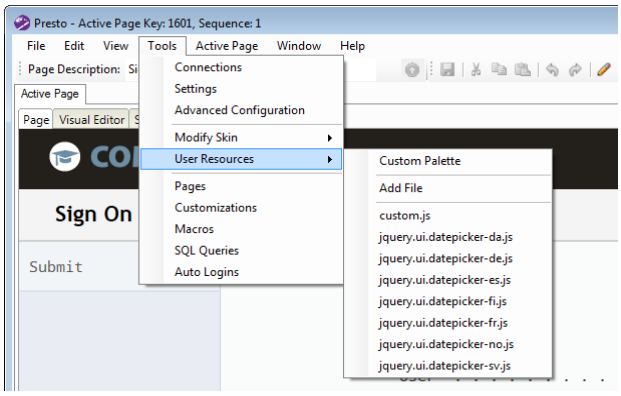


Figure 2-10. Opening custom.js in the Presto Designer.

The file custom.js includes instructions for overriding the default English settings. As you can also see from the illustration above Presto ships with several language files in the User Resources directory, so if your target language is one of these, you don't have to download anything additional:

Language	Code
German	de
Spanish	es
French	fr
Danish	da
Finnish	fi
Norwegian	no
Swedish	sv

Follow the instructions in the comments in your custom.js file to enable the appropriate language file. If required, you can also review or update the language file via the menu option shown in Figure 2-10.

Step 11: Additional Installation Considerations

Below are some additional considerations that may be relevant to you, depending on the type of applications you intend to build.

Demo Watermark on SmartCharts

As of Presto 6.6, the SmartCharts have a demo watermark 'SmartCharts Demo' on all the charts. This watermark is displayed on both the paid and trial versions of Presto:

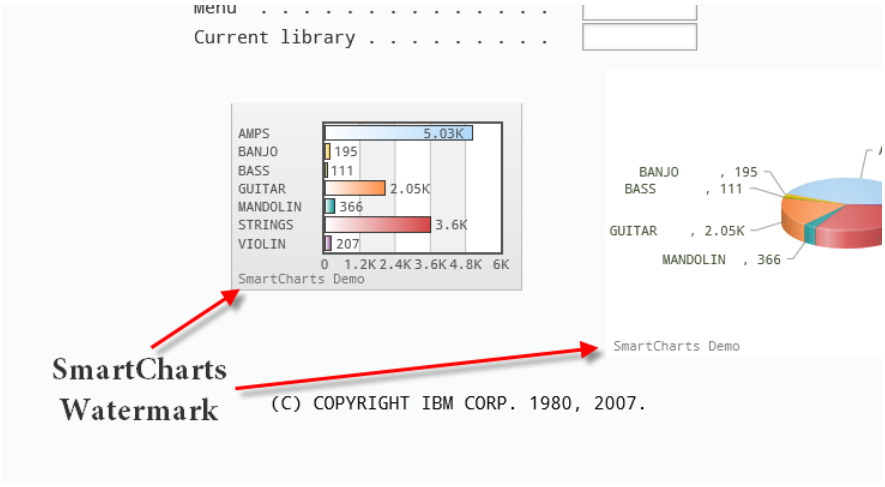


Figure 2-11. SmartCharts Demo Watermark.

In order to remove the watermark, you'll need to replace the demo version of SmartCharts with their commercial version. Contact Technical Support to get the SmartCharts commercial version that has no demo watermark.

Once the SmartCharts are replaced, you should no longer see the demo watermark appear on your charts.

Moving On

After you finish installing the software we recommend that you continue with the online tutorial at this address:

```
http://www.myfreschesolutions.com/nexus/apps/PRTUTRL.pgm
```



If you have any difficulties or questions, please contact our Technical Support team at 250-655-1766 [7:30 - 5:00 PST], or by email at support@freschesolutions.com. We pride ourselves in providing excellent support and we will be happy to help you out.

During business hours calls are picked up immediately by knowledgeable support reps who can answer your questions on the spot!

If you want to try your hand at troubleshooting any issues you are running into, you can also check “Advanced Technical Topics & Troubleshooting” on page 403.

Section 3 - Presto Reference

CHAPTER 6 - WORKING WITH THE PRESTO DESIGNER 33
Presto Out of the Box 33
Understanding the Range of Customization Features in Presto 33
Summary 35
Launching the Presto Designer 36
Configuring Your Environment(s) 37
Connection Settings 37
Server Settings 38
Detection Settings 40
Global Settings 42
IDE Settings 48
Skins 49
Advanced Configuration Options 53
Using the Presto Designer 63
Editing the Page Description 63
Testing Pages in a Browser 64
Using the Find Window 65
Changing the Layout of the Presto Designer 66
The Active Page Tab 67
The Screen View 67
Working with Page and Global Identities 68
Device Preview Dropdown 70
Working with Custom Files 71
Support for Saving Files Locally 71
Managing Pages and Customizations 73
The Pages Tab 73
Working with Customizations 74

Attaching Custom HTML, JavaScript or Transformations to a Page 75
Creating and Working with SQL Queries 77
Preview Query Results 79
General Considerations for Working with SQL Queries 80
Using SQL Queries in Custom HTML 80
Support for Dynamic Library and File Names in SQL 81
Using jQuery Selectors for SQL Parameters 82
Recording and Playing Macros 84
Recording a Macro in the Presto Designer 84
Replaying Macros in the Presto Designer 85
Playing a Macro from a Page 85
Managing and Editing Macros 86
Creating and Working with Auto Logins 88
Technical Requirements 88
Creating an Auto Login 89
How to Use Auto Logins 90
More About the Auto Login Template Shipped with Presto 92
Spool File Listings 93
Technical Details 95
Adding a Site Favicon 96
Creating a New File 97
Rotate Orientation of Display Mode 98
Access PC Applications from Your Presto Site 99
Requirements 99
How to Use this Feature 99
Dual Maintenance Management 105
How does it work? 105
Working with Display-File Changes 106
Page Up and Down Buttons 107
Submit, Cancel, Exit Buttons 108
Window and Table Text 109
Unique Naming of Customizations 110
Renaming a Customization 110
Naming Conflicts 111
Supported Characters 111
Customization Not Found 112

Export Customizations without the Contents 112
Considerations for Upgrading and Existing Customizations 113
Auto Detection Settings and Identities 114
Understanding Your Detection Settings 114
Why Use 'Auto' Detection Settings? 116
How Auto Panel Detection Settings Work 116
Window Detection 117
Potential Trouble Spots 117
Troubleshooting Page Identity Problems 119
Using Panel Identities to ID Screen Variations 122
Using Global Level Identities 124
Pages that Match More Than One Global Identity 132
Disable Page Identities 133
A Final Word of Caution 134

CHAPTER 7 - CUSTOMIZING WITH THE PAGE PROPERTIES 135

Working with Page Properties 136
Page Properties 137
Function Key Type & Text Options 137
Detect Menu Links 138
Adjust Last Row 139
Auto Formatting 139
Bypass Output Only Screen 140
Polling Interval (Seconds) 140
Detect Subfile Options 141
Hide Subfile Options 141
Table Detection and Custom Tables 143
Creating and Working with Custom Tables 144
Setting the Table Perimeter 146
Setting the Table Header 146
Adding Table Columns 146
Reviewing Your Changes 147
Working with Windows 147
Working with Window Options 148

CHAPTER 8 - GLOBAL RULES FOR PAGE OPTIONS 150

Global Rules for Page Options Manager 151

Working with Global Rules for Page Options 152
Anatomy of a Matching Criterion 154
How to create A Global Rule 155
Migrating Global Rules for Page Options 156

CHAPTER 9 - THE PRESTO VISUAL EDITOR 157

Using the Visual Editor 157
Selecting Page Elements 159
Moving Elements 160
Aligning Multiple Elements to a Common Baseline 161
Working with Element Properties 163
Reviewing, Resetting, Importing and Exporting Properties 163
Setting the Element Type 166
Transformation List Filtering 168
Resizing Elements 169
Example: Shortening Text Fields Using the Element Properties 169
Example: Using the Element Handle and Edges 170
Inserting New Elements 172
Selecting Multiple Items (Group Selection) 174
Editing Tables 175
Using Screen Values in the Visual Editor 176
Example 1: Using Screen Values as Autocomplete Parameters 176
Visual Editor Insert Elements Palette 179
Adding Custom Palette Entries 180
Reattach Transformations to Moved Elements 182
Copy Transformations between Pages 183

CHAPTER 10 - GENERAL TRANSFORMATION PROPERTIES 186

Changing Colors 187
Selecting a File 188
Events 189
Example: Triggering an Alert When a User Mouses Over an Element 189
Field Settings 191
HTML5 Input Types 193
The Class Attribute 194
Identification Properties 195
Padding Properties 196

Position Properties 197
Text Properties 198
Copy Attributes 199

CHAPTER 11 - VISUAL EDITOR TRANSFORMATIONS 200

Autocomplete 201
Switching a Field to an Autocomplete 201
Setting a Minimum Number of Characters to Enter Before Querying 202
Creating and Attaching a Query 203
Query Examples 205
jQuery Autocomplete Events 208
Block Container (div) 210
Adding Block Containers and Using the layer_N Class 210
Button 214
Function Key 215
Check Boxes 218
Data Export 219
Adding the Element to the Screen 219
Data Export Field Properties 219
Data Source Properties 220
Defining the Data Source SQL 221
Date Picker Field 223
Set Datepicker Options Based on Browser Language 225
Drop Down List 226
Data Source Options 226
Element Settings 226
Static Example: Returning the Displayed Value 227
Static Example: Displaying a Description for a Returned Code 228
Dynamic Example: Using SQL Queries to Generate Lists 229
Adding Static Values to Dynamic Lists 232
Blank Option for Dropdown 233
Google® Maps 235
Adding Markers 237
Linking a Map Snapshot to a Full Featured Google Map 238
Icon 239
iFrame 240

Image 241
Inline Container (span) 242
Label 243
Link 244
Password Field 245
Radio Buttons Transformation 246
Switching a Field to Radio Buttons 246
Creating a Static List of Options 247
Setting Dynamic Options 248
Signature Pad 250
SmartCharts 253
Chart Properties 253
Spinner 255
Spinner Properties 255
Tabs 257
Table 262
Text Area 263
Text Field 264

CHAPTER 12 - GLOBAL TRANSFORMATIONS 265
Global Transformation Manager 265
Adding a new Global Transformation from the Visual Editor 273
Migrating Global Transformations 273
Considerations 277
One Transform Event 278

CHAPTER 13 - ENHANCING YOUR PAGES WITH JAVASCRIPT 280
Adding Simple JavaScript to Presto Pages 281
Where To Create Your Customizations 281
Presto JavaScript Snippets 281
Hello World 283
Page HTML Visible via Dockable Window 284
Example: Adding a New Command Key 286
Adding a New Drop Down List of Commands 288
Adding an Event to a Drop Down Box 289
Adding Features to Your Datepickers 291
Using a Script to Add Dynamic Images to a Page 294

Adding SmartCharts with Custom JavaScript 296
Building a Dynamic Chart 296
Building a Chart Using Static Data 297
Building a Chart Using Screen Values 298
Support for Retrieving Dynamic Values from the Screen 299
Enhancing Autocompletes Manually 301
Using an External Web Program or Service to get Data for a Drop Down Box 303
Using the File custom.js 307
Datepicker Localization 307
Enabling an Alarm Sound 307
Turning Off the Message Waiting Indicator 307
Suggested Structure for Additional custom.js Enhancements 307
Add Key Mapping 308
Using a Transformation to Upload Files 309
Retrieve Job Info 310
SentRequest Event 311
PlayMacro Event 312

CHAPTER 14 - CREATING AND USING CUSTOM HTML 313
Creating New Custom HTML Entries 314
The Apprentice's Guide to Editing the HTML 316
Review the Basic HTML 316
Other Basic Panel Elements 318
The Presto Screen, Field and Attr [Attribute] Tags 320
Using Screen, Field and Attribute Tags 322
Generating Dynamic Screen Tags for All Text Constants 324
Considerations For Using Custom HTML 325
Optimizing the Initial HTML 325
Managing Static Text vs. Output-Only Fields in Custom HTML 326
Adding Green Screen Content to Your HTML 329

CHAPTER 15 - WORKING WITH YOUR PRESTO SKINS 331
About Presto Skins 331
Skin Support for Mobile Devices 332
Using jQuery Mobile 334

CHAPTER 16 - USING PRESTO WITH RPG OA 335
How it Works 335

Considerations 336
Important Performance Consideration for Upgraders 337
RPG Program Requirements 337
Customizing RPG OA Programs with Presto 339
About the jqGrid jQuery Grid Plugin 341
The Presto OA Grid API 342
Presto.Oa.Grid.GetField('FieldName') 342
Presto.Oa.Grid.SetSize(width, [height]) 342
Presto.Oa.Grid.SetSortable(bool) 342
Presto.SetScreenTitle('Title') 342
Example 342
Demo OA Program 343
CHAPTER 17 - THE PRESTO SKIN EDITOR 344
About the Skin Editor 344
Using the Skin Editor 344
Using Preset Themes 347
Making Changes to Preset Themes 348
Changing the Skin Logo 350
Color Options in the Skin Editor 353
Logo Options in the Skin Editor 353
CHAPTER 18 - WORKING WITH MOBILE CUSTOMIZATIONS 354
About the Mobile Customizations 354
How Mobile Mode Works 354
Enabling Mobile Customizations 355
Previewing Your Mobile-Specific Customizations 357
Page Customizations 359
CHAPTER 19 - USING THE RESPONSIVE VISUAL EDITOR 360
About the Responsive Editor 360
Understanding the Responsive User Interface 361
Activating Responsive Features 363
Change the Width of the Grid 363
Placing Elements in the Grid 364
Placing an Element: Drag & Drop Example 364
Position Properties 364
Resizing Elements 365

Saving Visual Editor Changes 366
Adding Responsive Grid 366
Adding a Cell 367
Adding a Row 368
Merging Cells 370
How to Select a Row 371
Removing a Cell 371
Removing a Row 372
Splitting a Cell 373
Visible On Property 373
Hidden On Property 374
Placing an Element: Highlighting 374
About Creating Responsive Grids 375
Considerations 375
Additional Features 375
Responsive CSS Classes 376
Example 376
Example 376

CHAPTER 6

Chapter 6 - Working with the Presto Designer

Presto includes two main components:

1. A run-time component that intercepts the 5250 datastream (or RPG program output delivered to the OA handler) and dynamically translates and reshapes that to run as a web page.
2. A Windows-based design tool for transforming and modernizing your screens.

The first component works automatically once the software is installed. The second component, the Presto Designer, is described in the following chapters.

Presto Out of the Box

Presto starts working for you as soon as it's out of the box. Immediately after installing the software and starting the HTTP server instance you can type the default Presto URL into your browser, sign on to your IBM i via the browser session, and start using your green screen applications there.

Once you install Presto start by taking a quick tour of your applications (running them in your browser!) to get a general sense of how close the software's default screen interpretation settings conform to your own design standards, and whether you need to adjust any of the global settings that apply to your entire environment.

Once you've set those, you're ready to start adding custom features. These range from very simple adjustments (should the function keys be displayed as buttons or text links?) to the addition of entirely new elements (drop down lists, images, interactive maps, autocompletes, data download buttons, etc.). The next chapters describe how to make these changes.

Understanding the Range of Customization Features in Presto

You will normally start the process of customizing your screens for one or both of these reasons:

- To add new web features that don't exist in the original green screen application.
- To adjust the way Presto renders pages by default.

Regardless of why you need to customize a page, the process is the same. With the Presto design tool you can make changes that will apply to a single page or to every screen in your environment.

The next few sections describe the levels at which you can customize your applications with Presto.

Custom Skins

Presto allows you to customize the application 'skin' used in each environment you install. Skins roughly correspond to style sheets (e.g. CSS files) plus a couple of included HTML files (a banner and a footer). With your skins you can set design standards affecting every program and page in your installation.

You can create custom skins to change the heading(s) presented above and below your pages, as well as to control how your table borders, fonts, text fields and screen colors are rendered.

The skin is the first option you set for an environment. Skins are discussed in of the reference section of this guide.

Global Settings

Each copy of Presto you install on your system (and you can install as many as you need!) has its own global settings. You can adjust these settings to customize features such as the handling of menus, subfile options, function keys and error messages.

In the Designer you can chose the global settings that work best for your application. Changes are instantly rippled through to how your users see their programs.

For instance, if you have applications with widely divergent design standards (perhaps one you've bought and another you've developed in-house), you can run multiple copies of Presto on different ports and tune your global settings for each application.

See the section "Separate Presto Installation and HTTP Server Instances" on page 35 below for more details.

Page Setting, and Custom Tables and Windows

Most environment level settings can also be overridden on individual pages. In addition, some of your screens will benefit from custom windows and tables, where you override the dimensions and other attributes of these containers.

The features for working with page settings and building custom windows and tables are described in "Customizing with the Page Properties" on page 135

Using the Visual Editor

The Visual Editor tools are very useful for changes ranging from simply resizing or otherwise changing individual screen elements (making fields shorter or changing the style for page headings) right up to replacing plain fields and text with elements like static dropdown boxes, date pickers, autocomplete fields and charts. You can also use the visual editor to insert new, custom screen elements.

These features are described extensively in the tutorial as well as in "The Presto Visual Editor" on page 157.

Using Custom Scripts

You can use custom JavaScript to enhance individual pages, or make changes across your entire site.

The tutorial section of this guide includes two chapters with several illustrations using custom scripts, and the reference section includes additional details.

Creating Custom HTML

In some cases you may want to completely redesign some of your pages from the ground up, using Presto to deliver your data. With the Presto Designer you can create custom HTML where Presto allows you to combine the fields (and any text you want to keep) from your original programs with new JavaScript, HTML or embedded web programs.

These features are described beginning in , Creating and Using Custom HTML.

Changing Your Original Green Screen Program

Finally, it's also worth noting that while Presto doesn't often *require* changes on the green screen side, in practice you may find that some types of changes and improvements are most easily accomplished starting by changing your original programs.

For example, if you have a subfile with a less-than-ideal layout or several flat panels with inconsistent or duplicate headings, you could use the tools in the Presto Designer to fix these problems. But you may prefer to just make some minor layout changes to the original programs instead.

Separate Presto Installation and HTTP Server Instances

If you have multiple applications to modernize, built with radically different design standards, it's possible that one set of Presto defaults that works very well for one application or group of programs is no good for another. In this case, rather than individually customizing all the panels in one or the other set of programs, you can install completely separate Presto environments where you adjust your settings and skins to suit a subset of your programs.

, Additional Presto Installation Options, describes how to install more than one copy of the Presto server components and configure your designer to work with each one.

Summary

Presto is a development tool designed to handle many different needs and modernization approaches. Some customers use Presto to deliver their IBM i applications in a browser almost entirely using the out-of-the-box settings. On the other end of the continuum, other customers rely heavily on custom HTML, preferring to hand-code almost all their HTML, CSS and JavaScript to create unique and highly customized applications that you'd never even guess were green screen programs under the covers.

The tool adapts to one approach as readily as to the other.

Most customers are using a combination of several different strategies to achieve their goals - Presto's flexibility should allow you to achieve similar success in your own project. The next several chapters of this guide provide detailed reference information on all the design options available.

Launching the Presto Designer

Launch the Presto Designer from Start>Programs>Presto>Presto. The executable is typically at C:\Program Files (x86)\ESDI\Presto\Presto.exe.

If you followed the installation instructions in this guide Presto is already configured to connect to your server. If not, review the instructions beginning in Step 2 of the installation instructions ().

Below is the Presto Designer, when you first launch it and have signed on to establish a connection between the Designer and the server:

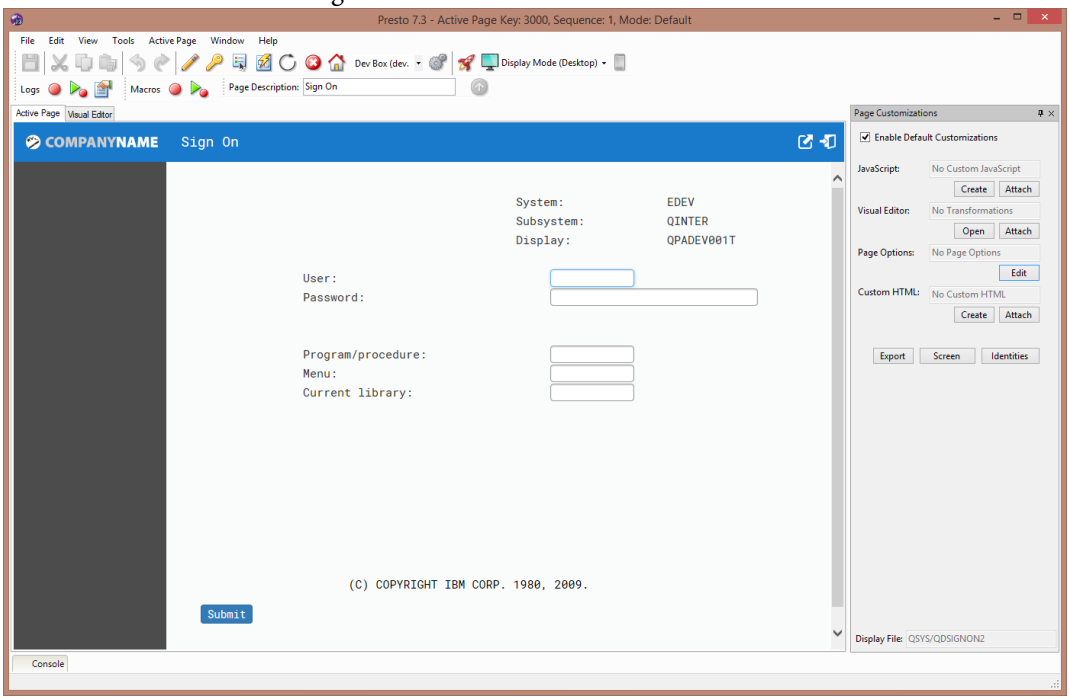


Figure 2-1. The Presto Designer upon installation and first connection.

In this illustration, Presto has already been configured to connect to a default system, and the main area of the design screen is showing the IBM i Sign On screen for that system.

The image and Presto logo at the top of the page come from one of the default skins shipped with the software, named 'Nightfall'.

The content in the main body of the design tool corresponds to what your users will see when they connect to your IBM i through Presto. Within the Designer you also have a variety of options on the right that you can use to work with how your applications look and run.

Configuring Your Environment(s)

Use the Presto Designer Tools menu options to set up your environments. An 'environment' is a unique Presto server side install with its associated Apache HTTP server instance.



Presto includes a number of settings that are applied at the environment level. For instance, you can only have one skin active per environment. There are also other automated settings that you may want to turn on or off for entire applications, rather than by individual panels.

When you select the Tools>Connections option Presto displays this window:

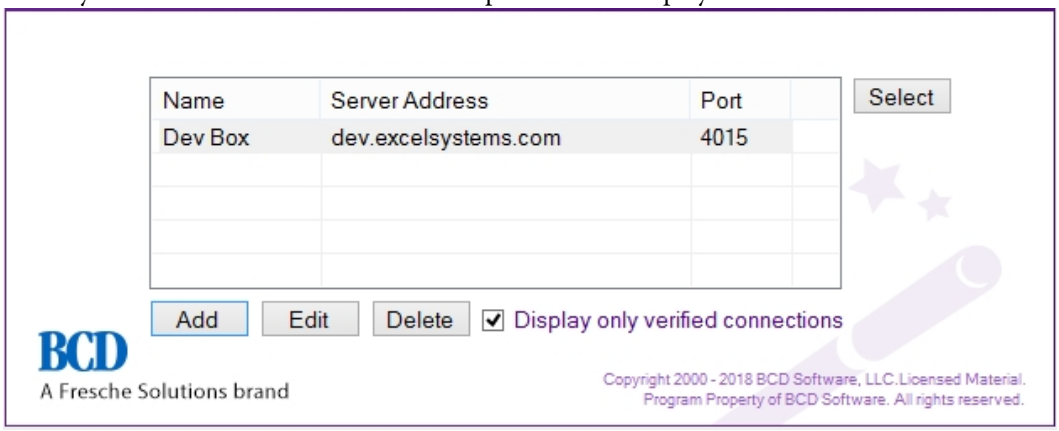


Figure 2-2. Presto connections (or environments).

In this illustration we have already defined an environment. Use the Add and Change buttons to create and configure your environment(s).

Connection Settings

These are the options Presto prompts you for when you add or change a connection:

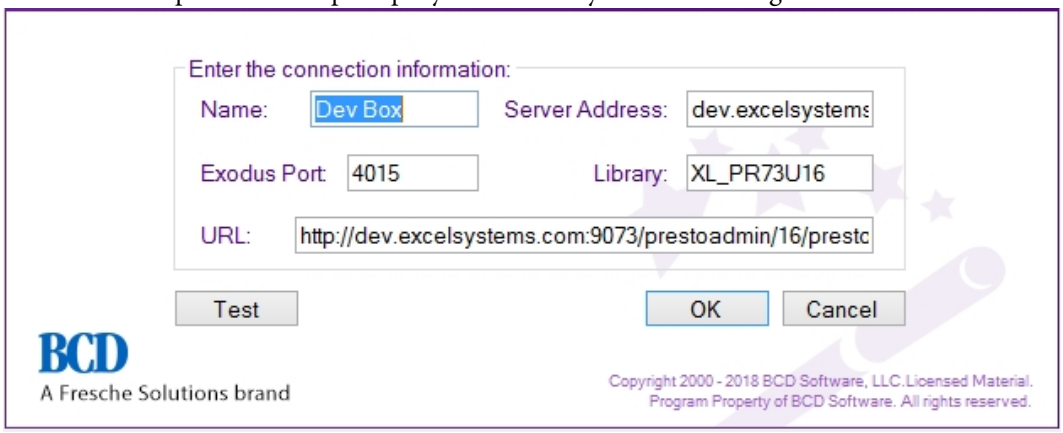


Figure 2-3. Presto environment attributes.

Connection Name
This description is displayed when you establish connections and from the drop down list in the Designer.

Server Address
Enter your server's domain name or IP address.

Exodus Port
The default port of 4015 for your Exodus55 connection usually works.



If you need to use a port other than 4015 here, you must also run the `XL_EX55/ZCFGTCP` command to change your server side communication (press F4 to prompt the command). After making that change, restart the `EXODUS55` subsystem.

Library
The default Presto library is `XL_PRESTO`.

During the install you are prompted for the library to install to. You can enter a different library name to run multiple versions of the software. You must use the same library name above as you as specified on your server side installation.

See the note on the previous page for further details on this feature.

URL
This is the URL the Designer will use to connect to your IBM i and edit your applications. Here is an example of a valid URL:

```
http://192.168.0.111:8022/prestoadmin/presto
```

This begins with your internal server IP address followed by the port number of the default Presto HTTP server instance (port 8022 in this case). Following that is the path to Presto, which initiates a new session.

If you have installed a second copy of Presto to a different library and have another HTTP server instance running on a different port, specify that port in this path. For more information, please read , Additional Presto Installation Options.

Testing your Connection
When you click the Test button Presto attempts to connect to the server using the settings provided, and then pops up a confirmation message.

Click OK to save your changes.

Server Settings

Under the Tools menu you'll also see an option titled Settings. The Server Settings here correspond to the environment to which you are currently connected. These are the attributes associated with an environment:

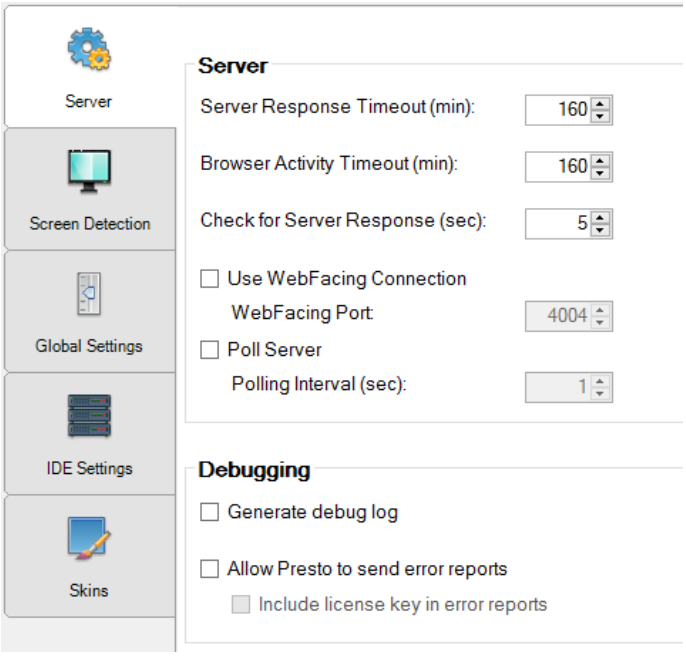


Figure 2-4. Defining your server connection details.

Server Response Timeout (Minutes)

This is how long Presto will wait for a response from your system before issuing a timeout message.

Browser Activity Timeout (Minutes)

This is how many minutes a Presto session will remain connected without activity. This value is important if on the green screen your users may be accustomed to leaving their sessions open & inactive for hours at a time. In the Presto environment, sessions will time out if they are left inactive for too long.

If your users frequently report their sessions being terminated you can increase this time out value to a setting that works better for your user's typical usage of their applications. Or you can tell your users to end their sessions if they're not going to use them!

Check for Server Response (Seconds)

Use this option to mitigate against multiple, long-running jobs locking up your available webserver jobs, blocking all additional requests. For example, if in your environment it's possible for users to initiate large interactive reports that take some time to return control, if enough people submit these reports at the same time, they could lock up all the jobs available for other requests, causing slow response times to other users, or even session errors.

Set this value to a low number (5 or 10 seconds) to allow those jobs to open up for other requests at that interval.

Use WebFacing Connection

If you have the WebSphere V6R0 (or higher) Application Server installed you can enable Presto's option to use WebFacing. By using WebFacing you can avoid using interactive resources for your Presto sessions.

This feature is discussed in more detail in the section "Using WebFacing with Presto" on page 397.

WebFacing Port

This port is used for the WebFacing connection. You only need a value here if you are using WebFacing in this environment.

Poll Server/Polling Interval

This option is documented in the section “Activating Polling Support” on page 408, Advanced Technical Topics & Troubleshooting.

Generate Debug Log

When you check this option the jobs in QUSRWRK produce logs of debug data. These logs can quickly get very large, so do not turn this option on unless instructed to do so by Fresche Technical Support. You can find the log files in your IFS, at /root/esdi/env_name/log folder, where ‘env_name’ is the name of the environment you’re working in. The default is XL_PRESTO.

When debug is on, this is shown in the title bar:

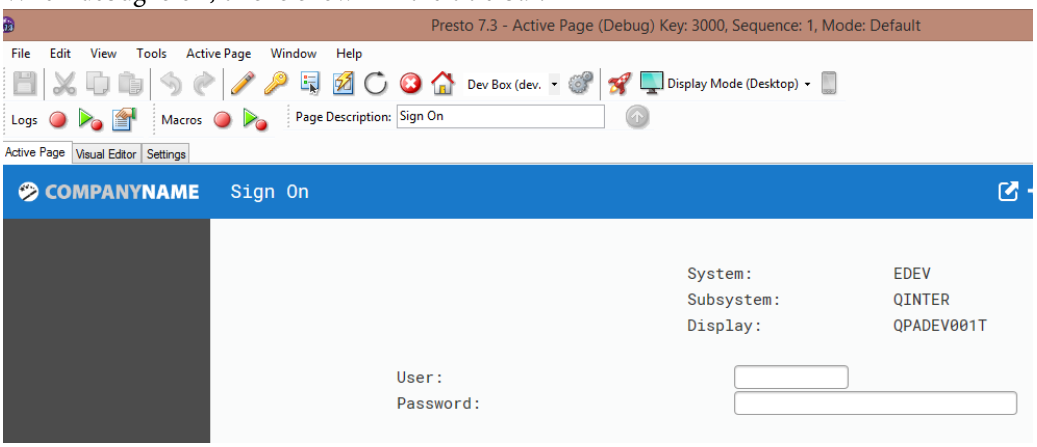


Figure 2-5. Running the Presto Designer in Debug mode.

End and restart the Presto Designer for a change to this option to take effect.

You can change the level of detail of the debug logs by setting the 'debuglevel' configuration value in Advanced Configuration (Tools>Advanced Configuration).

Allow Presto to Send Error Reports

When you check this option, Presto can send crash reports to our Technical Support.

Detection Settings

The Detection Settings include these options:

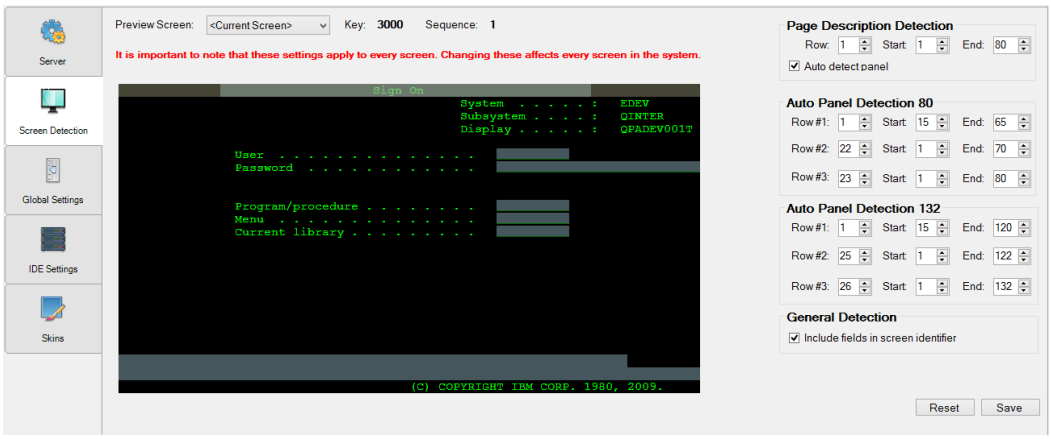


Figure 2-6. Detection settings.

Page Description Detection

Page descriptions are an important component of your Presto customization project. Inconsistent or repeated page names make it difficult to manage your customizations. For this reason we recommend that you take the time to ensure that all the pages you touch with Presto have meaningful descriptions.

Presto normally pulls page descriptions from the first line of your screens. If your applications typically have titles or descriptions at a different location, you can change the default setting in your Detection Settings.

Auto Detect Panel Description

Presto by default uses some 'smart' logic to extract the page description from the row section you specify in the detection settings. For example, certain keywords are ignored. If this logic doesn't generate good results for you (e.g. the default page descriptions aren't consistently useful without correcting them by hand), you can try unchecking this option and just use a hard coded row/column/length combination to build your page descriptions.

Auto Panel Detection [80 and 132]

Presto uses portions of text on your screens to uniquely identify them in order to correctly apply the custom features you add over your programs. The portions of the screen identified by these options must remain reasonably static (i.e. not include time and date fields, for example, which change every time you run the program). If you include variable data within these boundaries, every time the program runs Presto will think it's a new screen, and not apply your customizations.

80 Column Row #1, Start & End

Presto is by default set to expect to find a program title in the area between columns 15 and 65 of row 1. If you place your program title in a different area, or this range of columns typically includes variable data, you can change these settings to be a better match for your program title, or any unique information that you have in the top row.



Some customers have adjusted their applications specifically to be web-enabled by Presto, by adding a unique screen ID to every unique green screen page. This is one way to simplify your auto-detection settings, and ensure your customizations are always applied to exactly the right screens.

80 Column Rows #2 & #3, Start & End

On lines 22 and 23 Presto by default expects to find a particular set of function keys. Presto uses these, along with the program title and other screen characteristics, to uniquely identify each panel in your applications.

If columns 1 to 70 of row 22, or any spot on row 23, includes variable data, you can adjust these settings to exclude that data.

You can also switch your settings to use different rows.

132 Column Row #1, Start & End

Presto expects to find a program title in the area between columns 15 and 120 of row 1. If you place your program title in a different area, or this range of columns typically includes variable data, you can change these settings.

132 Column Rows #2 & #3, Start & End

By default, Presto expects to find a particular set of function keys on lines 25 and 26. Presto uses these, along with the program title and other screen characteristics, to uniquely identify each panel in your applications.

If columns 1 to 122 of row 25, or any spot on row 26, includes variable data, you can adjust these settings to exclude that data.

Include Fields in Screen Identifier

In some cases, your applications may include many screens with conditionally displayed fields. Because Presto uses the fields displayed on the screen as a part of its criteria for determining if a saved, custom HTML entry 'matches' the user's active session, you may encounter situations where the customizations aren't applied to a panel because a particular field is either present or missing.

If you suspect that conditionally displayed fields are consistently breaking links to your customizations, you can check this option for your environment.

Normally, you shouldn't click this option until you have ruled out (in consultation with Technical Support or unless you are very certain you understand what's going wrong with your panel) all other possibilities for custom HTML not being applied correctly.



This option, and the Auto Panel Detection option below are very important. Before you get too far into customizing your site with Presto it is important that you read "Auto Detection Settings and Identities" on page 114 and make sure you understand how Presto uses these critical settings. Changing them after you begin your development can create a lot of extra work for you.

Global Settings

Click the third tab in the Settings window to work with these Global settings:

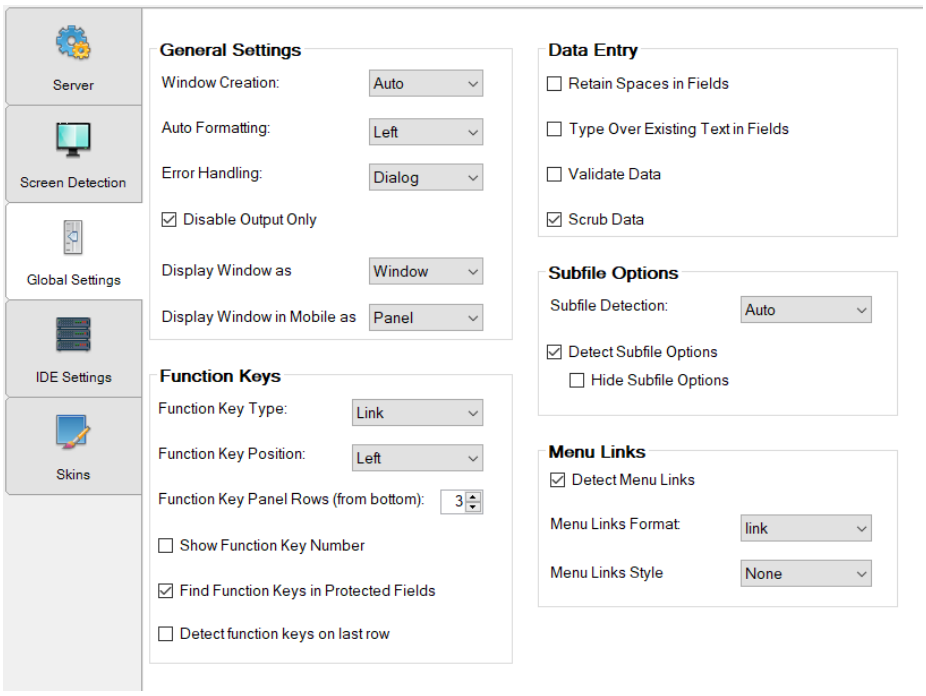


Figure 2-7. The global settings for a program or application.

Click OK to make any changes, or Cancel to clear your changes.



Changes you make here immediately affect every page presented by Presto in this environment. It is best to make all your environment changes before making applications available in Presto to end users, or use a test environment to verify the outcome of your change.

Below are descriptions of the options.

Window Creation

This environment option determines the default setting for window creation on your screens.

- **Auto.** If you set the option to Auto, Presto will automatically scan your panels and build an HTML window at the most likely location.
- **None.** If after reviewing your application with Presto, you determine that the standard screen layouts in your application aren't usually compatible with the approach Presto takes to building windows automatically, you can change your environment default to None.

Regardless of the option you choose, you can still create new, hand-drawn windows on any of your screens.

This option is also available at the individual page level, and is further discussed in the section "Working with Windows" on page 147, Customizing with the Page Properties.

Auto Formatting

Use this option to set the screen automatic formatting options shipped with the software. Currently, this option removes trailing dots from field text, aligning the text either right or left.

For example, this string:

```
Customer Name . . . . . .
```

is converted to:

```
Customer Name:
```

If you select the option 'Right', the text is aligned to the right, adjacent to the field. If you select 'Left' the text is aligned to the left edge of the page.

This option is also available at the individual page level, and is further discussed in the section "Adjust Last Row" on page 139, Customizing with the Page Properties.

Error Handling

Use this option to set how Presto handles system errors that in a 5250 display lock up the screen. These are the options:

- **Default.** Use this option to display errors in a message subfile and lock the keyboard if the user presses an invalid key (similar to 5250 sessions).
- **Dialog.** Pick this option to present the error messages in a dialog box.
- **Ignore.** Select this option to have Presto ignore any system errors.

This feature was added at the request of a customer who felt that their web users would be unaccustomed to behaviors like keyboards locking up.

Disable Output Only Screens

This option works in conjunction with 'Temporary stop of output only screens' option (Ctrl+F11). Use this feature to allow you to customize screens that normally pop up only for a second before advancing. Here is how you can use these options:

1. When this option is unchecked and the 'Temporary stop of output only screens' option is checked, Presto pauses for a few seconds on output only screens (allowing you enough time to customize them). This is for editing output screens.
2. When this option is unchecked and the 'Temporary stop output only screens' option is also unchecked, Presto displays the output screen only for a moment. This is for displaying output screens.
3. When this option is checked, the 'Temporary stop output only screens' option is not relevant.

You can set this option at the environment level (if you are is interested in working with any output only screen) or at the page level (when you want to work with a specific screen).

Detect Menu Links

Check this option to have Presto automatically convert menu options to hyperlinks. This option is turned on by default.

Menu Links Format

Use this option to set the default appearance of your menu links. The options include:

- **Link.** Use this option to render the menu options as links.
- **Button.** Select this option to render the menu options as clickable buttons.

Menu Links Style

This option is designed to be used in conjunction with link-style menu links. The options include:

- **None.** Use this option to not perform any styling on the menu links. They will work as clickable links.
- **Arrows.** Use this option to render an arrow beside the menu link.

Retain Spaces in Fields

This option allows you to set how your Presto pages should handle text fields. These are the effects of selecting this option:

Option Off	Option On
Clicking an empty field will always place the cursor at the start of the field.	Clicking an empty field will place the cursor at the exactly location of your cursor when you clicked.
Cursor is in 'insert' mode.	When you place the cursor in a field and start typing, it overwrites the text to the right, and at the end of a field automatically jumps to the next and begins overwriting that as well.

Type Over Existing Text in Fields

This option works similarly to the 'Insert' key on your keyboard. If you check the Type Over option, anything that a user types in a field that already contains text will overwrite the text to the right. At the end of the field, you will jump to the next field and begin overwriting that as well.

When this option is checked, a user can simply turn the Type Over mode off via the 'Insert' key on their keyboard. Every time a screen is loaded, however, the option will revert back to Type Over.

Note that when you check the Retain Spaces in Fields option in your global settings, the Type Over option is automatically turned on. In this case, the 'Insert' mode on your keyboard does not work.

Validate Data

Presto incorporates allows you to implement automatic server side validation on field types and attributes.

If a screen attempts to return data to the server that doesn't match the attributes the server program is expecting, the session is terminated and Presto issues the error:

```
Presto Error!

Abnormal Termination: Data has been passed back to the server of the wrong type. Please
contact your site administrator.
```

To enable this feature at the environment level check the 'Validate Data' option and save your change.

You can also enable/disable this feature for individual pages by editing the Page Properties.

Scrub Data

Presto includes a feature to strip characters from web page submissions which are likely to be indicators of an attempt to add bogus data. For example, if you have a public web page that allows customer to enter free format notes, someone may attempt to embed a URL to a bogus page in a way that lures other website users into clicking the link.

You can 'scrub' the following characters from submissions:

- <
- >
- &
- '
- "

Use the checkbox above to enable this feature at the environment level.

You can also enable/disable this feature for individual pages by editing the Page Properties.

Subfile Detection

Presto can attempt to automatically detect your subfiles and convert them to HTML tables. These tables have a number of nice features, including alternating row colors.

- **Auto.** Choose this option to enable subfile detection.
- **None.** Set this option to None to display subfile rows using the same standards as any screen text.

You will normally only select the None option if you find that typical subfiles in your application aren't compatible with the approach Presto takes to building tables automatically. In these cases you can turn off the automatic formatting and style the subfiles manually on individual pages.

The features for creating tables manually at the individual page level are discussed in the section "Table Detection and Custom Tables" on page 143, Customizing with the Page Properties.

Detect Subfile Options

When this option is enabled Presto automatically scans your subfile screens for text strings that look like subfile options, and makes them available when the user hovers overs the options field:

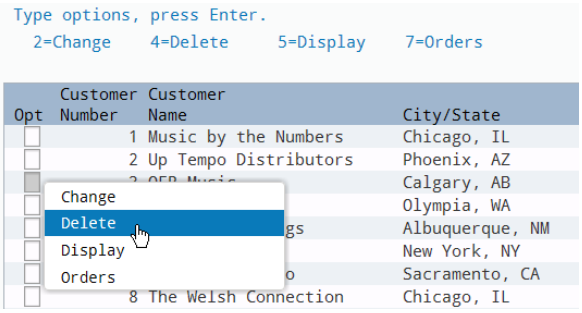


Figure 2-8. Outcome of automatically detecting subfile options.

In this illustration you can see that the program has 4 subfile options available. Presto has extracted the text from these and presenting the text when the cursor is over the options field.

You can adjust the size and location of this window in your stylesheet.

If you have a subfile that Presto has not detected automatically, but want to enable the flyout menu options illustrated above, please check KB#15078:

https://www.myfreschesolutions.com/kb?kbnum=15078

The Hide Subfile option described below allows you to hide the subfile options and only show them in the flyout menu.

Hide Subfile Options

This option allows allows you to automatically hide the subfile options that are detected on the subfile. The subfile options will be available to a user in the flyout menu.

This option is available at both the environment and page level.

Detect Everywhere

Use this option to detect function keys either everywhere or only inside of the function keys panel.

Function Key Type

Use this option (together with the other options below) to set the default appearance of your function keys. The options include:

- **None.** Use this option to not perform any conversion to buttons, leaving the function keys as non-clickable text entries on your pages. They will still work when users press them on their keyboards.
- **Button.** Select this option to render function keys as clickable buttons.
- **Links.** Use this option to render function keys as text hyperlinks.



Presto recognizes (and automatically transforms) your screen function keys based on criteria set in the regular expressions found in the configuration file config.bws. Please refer to the section “Customizing Function Key Text Recognition” on page 410 for additional information.

This option is also available at the page level.

Function Key Panel Rows (from bottom)

Use this option to set the number of rows (from the bottom of the screen) that Presto will scan for function keys. For example, if you use the default value of 3. Then on a 24x80 screen Presto will scan from line 22 to line 24.

Function Key Position

This option is designed to be used in conjunction with button-style function keys. These are the available choices:

- **Left.** When you first install Presto the function keys are added in a column on the left side of the page.
- **Right.** You can also display function keys in a column on the right.
- **Bottom.** To leave your function keys on the bottom of your pages, as they are in the green screen programs, select the option Default.

Show Function Key Number

By default Presto renders function keys as "F#=Action". If you don't think your users will need to know the function key numbers, you can uncheck this option to drop the "F#=" portion of the string.

Note that this is a cosmetic change to your screens. The actual keyboard function keys will continue to work regardless.



As previously mentioned, Presto recognizes (and automatically transforms) screen function keys based on criteria set in the regular expressions found in the configuration file config.bws. Please refer to the section "Customizing Function Key Text Recognition" on page 410 for additional information.

This option is also available at the page level.

Find Function Keys in Protected Fields

If your convention for function key text isn't distinctive or you have odd data in your files it's possible that Presto may believe it has found function keys embedded in your field values and display them as buttons.

Here is a glaring example of this:

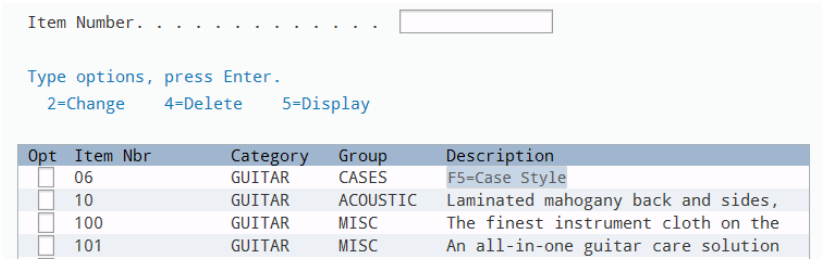


Figure 2-9. Finding function keys in strange and wonderful places.

In this illustration we just happen to have a guitar case in inventory with a description of "F5=Case Style". Because this text string matches our standard function key pattern Presto has rendered the text as a button.

If this type of thing happens in your applications you can uncheck the 'Find Function Keys in Protected Fields' option and try reloading the page.

This option is also available at the page level, in the Page Properties.

IDE Settings

Click the fourth tab to access the IDE Settings (Tools>Settings>IDE Settings tab):

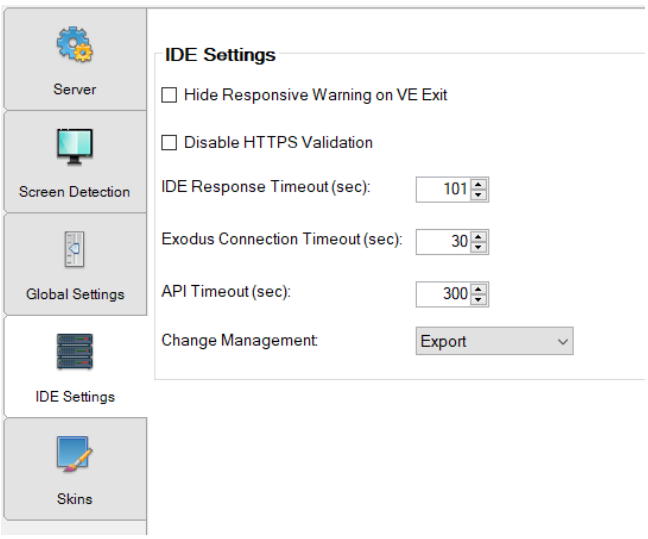


Figure 2-10. Selecting the IDE Settings.

API Timeout

This option allows you to configure the maximum wait time for http requests used for commands like Save, Export, and Import. The default timeout is set to 300 seconds (5 minutes). This setting is useful when exporting a large number of screens that take a long time to run.

Skins

An environment skin is a collection of attributes used to format the contents of your pages. Presto ships with several skins.

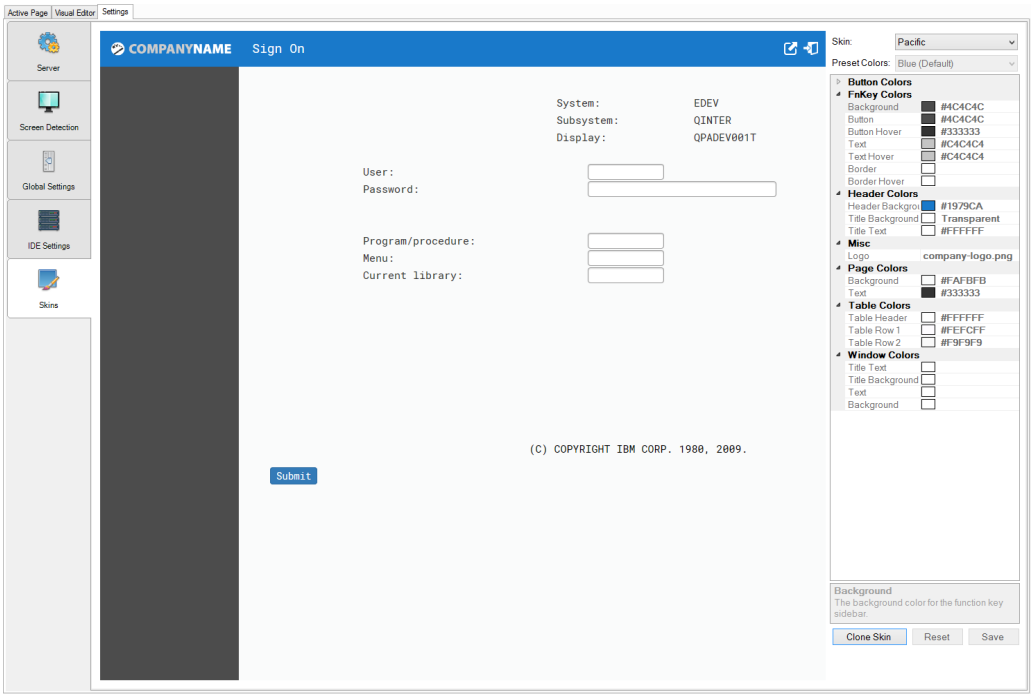


Figure 2-11. Selecting the Skin to use in the Presto designer.

You can create a new skin by cloning one of the existing skins by going to Tools>Modify Skin>Clone Skin.... You can then modify the copied entry. The tutorial has an example you can follow.

The skin will appear in the Presto Designer the next time you open it.



All the skins use absolute positioning and monospace fonts to make your rows and columns line up.

As you're getting started with Presto, it's a good idea to spend some time experimenting with the various skins, navigating around your own applications. If you create your own skin you can use one of the shipped ones as a starting point. Please refer to the tutorial section "Trying Different Environment Skins" in the online tutorial for some guidance on how to experiment with different skins to find the one most suited to your needs.

Skin Themes

Themes allow the end user to select a color scheme for their Presto session.

When the user logs in to Presto in their browser, they will see a circle icon which expands to a menu to choose among enabled themes.

The initial enabled themes will be Default and Dark. However, these themes can be configured within the Presto designer.

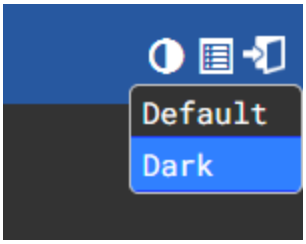


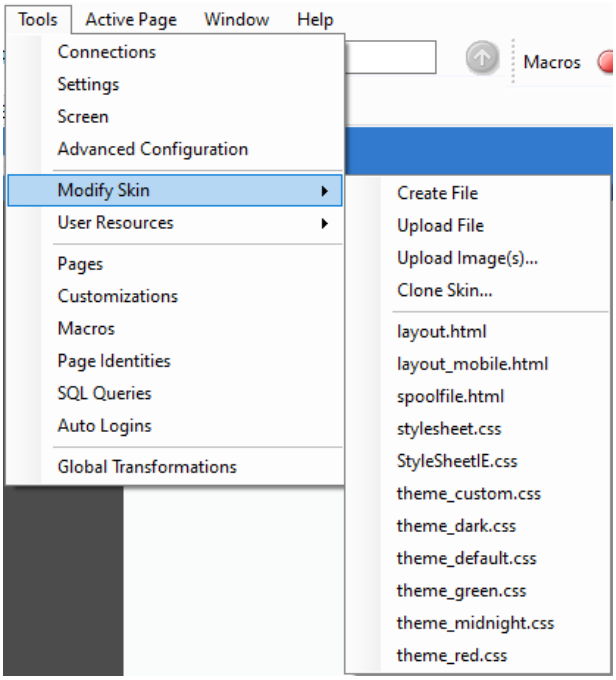
Figure 2-12. Choosing between the enabled themes.

Enabling the Themes

The list of themes available to the end-user is set in Tools>Advanced Configuration (config.bws). Locate the module with the name ThemeSelector.

```
<module name="ThemeSelector" enabled="1">
    <parameter name="midnight">Dark</parameter>
</module>
```

In the above module, there is one parameter element with a name of midnight. This name references one of the existing Themes that shipped with Presto. You can view the shipped Themes by accessing the Tools>Modify Skin menu in the Presto Designer:



The actual theme files will follow naming convention theme_themeName.css where themeName is what should be referenced in the parameter name.

In the above example for: `<parameter name="midnight">Dark</parameter>`, the theme is called theme_midnight.css.

The value 'Dark' is the option displayed in the menu that the end user can select. This value is arbitrary.

Adding a theme to the menu

Adding a theme to the menu is done by modifying the Advanced Configuration. Adding another `<parameter></parameter>` element will add that option to the menu.

For adding the red theme to a Presto environment would look like this:

```
<module name="ThemeSelector" enabled="1">
    <parameter name="midnight">Dark</parameter>
    <parameter name="red">Redness</parameter>
</module>
```

The end user would then see the 'Redness' option in their browser, and they can choose to use that color scheme during their browser session. See the following section on how to customize a theme.

Customizing Skin Themes

Presto allows you to customize multiple themes in one skin.

Customizing the theme is done by modifying a Preset Color in the Skin Editor (Tools>Settings>Skins). Presto will add the custom theme.

For example, modifying the 'red' theme creates 'custom_red' which looks like this:

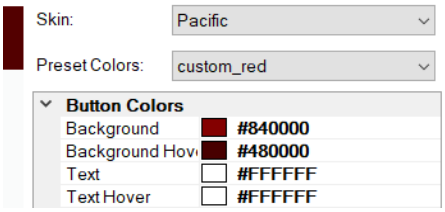


Figure 2-13. Customizing the red theme.

After saving the custom theme, you can access it under Tools>Modify Skin:

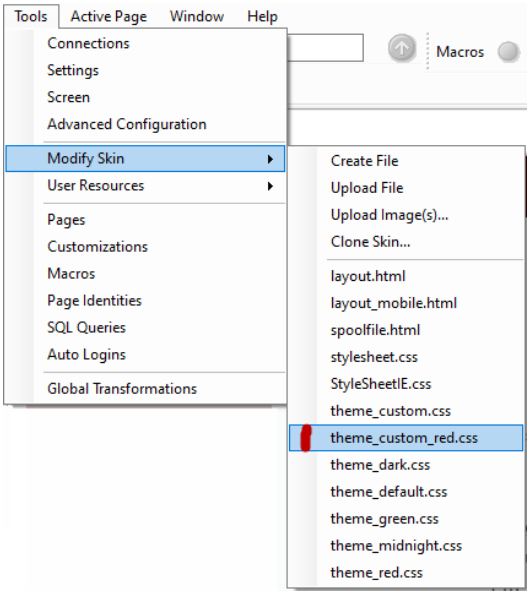


Figure 2-14. Viewing the custom theme file in the Presto designer.

Go to Tools>Advanced Configuration to add the custom theme to the list of themes available to the end user:

```
<module name="ThemeSelector" enabled="1">
      <parameter name="midnight">Dark</parameter>
      <parameter name="custom_red">Red</parameter>
</module>
```

When the user logs in to Presto in their browser, they can select the custom theme:

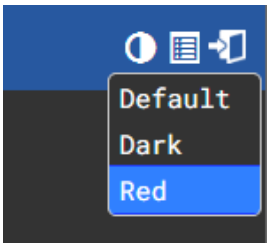


Figure 2-15. Selecting custom theme in the user's browser.

The end user can now select the custom theme 'Red' in their browser to use that color scheme during their Presto session.

Advanced Configuration Options

The Presto Designer includes a Tools>Advanced Configuration option which allows you to open and edit the internal file config.bws:

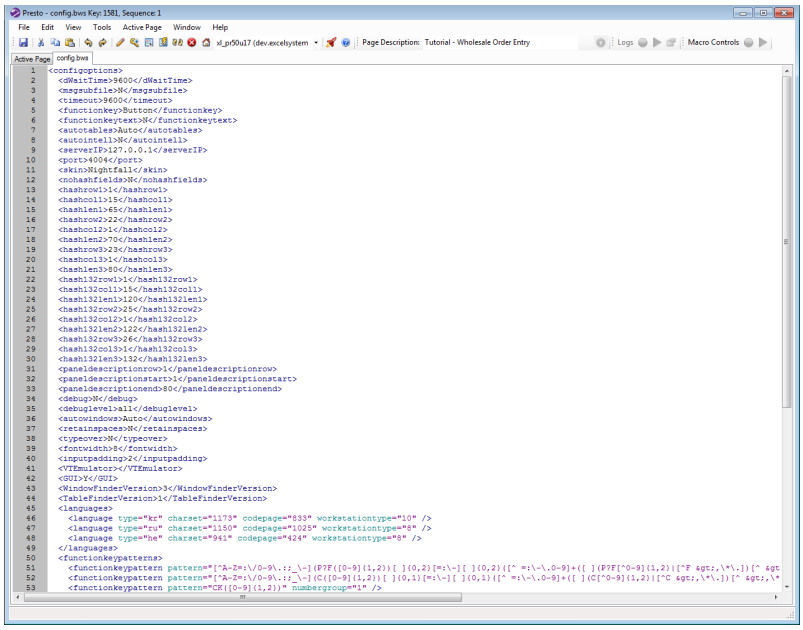


Figure 2-16. Opening and editing config.bws.



Unless you are very familiar with the software, you should only change these settings when instructed to do so by a Fresche Technical Support rep, a Knowledge Base article, or the User Guide.

Here is an overview of the available options.

Config Value	Description
StartupTimeout	The amount of time in seconds that Presto will wait for an answer from the server when it initially starts up. After this period Presto will stop loading.
SpoolFileTimeout	The amount of time in seconds that Presto will wait while downloading a Spool file. If the process to build and return the Spool file takes longer than this, Presto will return an error and the session will most likely be killed by the timeout. For most users, the default of 90s is more than enough, very large Spool files may require a higher value.
dWaitTime	Corresponds to Settings→Server→Server Response Timeout (converts from minutes to seconds)

Config Value	Description
	The time Presto waits for a response in the data queue. After this period Presto will cancel the request and display a timeout error message.
msgsubfile	No Message Subfile option. By default Presto treats line 24 as a message subfile. If you use programs that do NOT use line 24 in this way, you can enable this option so that Presto can correctly handle that line. If only some of your programs don't use line 24 as a message subfile, you should enable this option at the page level for those programs instead.
timeout	Corresponds to Settings→Server→Browser Activity Timeout (converts from minutes to seconds). Webfacing timeout and JavaScript Session timeout
functionkey	Corresponds to Settings→Environment→Function Key Type How function keys should be displayed (Button or Link)
functionkeytext	Corresponds to Settings→Environment→Show Function Key Number Option to display full text of a function key (Y/N)
autotables	Corresponds to Settings→Environment→Subfile Detection Tables auto-detection (Auto, None)
autointell	Corresponds to Settings→Environment→Auto Formatting Autoformatting option. Removes dots from between text and fields on screens, and adjusts the text right or left. (RIGHT, LEFT, None)
serverIP	WebFacing Server IP address.
port	Corresponds to Settings→Server→WebFacing Port WebFacing Server Port.
skin	Corresponds to Settings→Environment→Skin Selected skin.
nohashfields	Disable including fields in the hash (Y/N).
hashrow1..3	Use string from a given row in hash.
hashcol1..3	Use string from a given column in hash.
hashlen1..3	Use string of a given length in hash.
hash132row1..3	Use string from a given row in hash (132 col screens).
hash132col1..3	Use string from a given column in hash (132 col screens).
hash132en1..3	Use string of a given length in hash (132 col screens)
paneldescriptionrow	The row from which to draw the panel description.
paneldescriptionstart	The position at which to start looking for the panel description.

Config Value	Description
paneldescriptionend	The position at which to end looking for the panel description.
debug	Enable debugging (Y/N).
debuglevel	The detail level of the debug logs (what you want to debug). The available settings are critical, all, info, timing, error, or none. The default setting is 'critical'. This will mean only critical errors are logged in the debug log. If the setting is 'timing', Presto will reduce the amount of information to the debug log. Only logging basic events such as sending/waiting on the CGI and VT provider. Debug logs are enabled in Settings>Server>Generate debug log. Debug logs can be viewed in Help>Manage Server logs.
autowindows	Auto window detection setting (Auto/None). This option corresponds to the Window Creation option in your Global Settings.
retainspaces	Input fields filled with spaces in order to simulate ability to place cursor in any place of the field (Y/N).
typeover	Type over old text (Y/N).
fontwidth	8 (deprecated) This setting is now determined by CSS.
inputpadding	2 (deprecated) This setting is now determined by CSS.
VTEmulator	This options is related to the WebFacing Connection settings. (Tools>Settings>Server Settings).
GUI	This option allows you enable the ability of the terminal to use enhanced UI items, such at GUI windows, mouse functions, drop downs, etc.
WindowFinderVersion	Controls the logic for auto detecting windows. The Window Finder logic is versioned to allow you to control whether you enable a change on upgrade.
TableFinderVersion	Controls the logic for auto detecting tables. The Table Finder logic is versioned to allow you to control whether you enable a change on upgrade.
languages	Special language settings. <language type="VAL" charset="VAL" codepage="VAL" workstationtype="VAL" />

Config Value	Description
• type	These attributes are all system specific. To determine the correct values, run DSPDEVD and enter the Display value. On the Character Identifier line, the 'charset' is the first number and the 'codepage' is the second. To find the correct workstation type, try finding your OS version here, scroll down to the 'Supported Workstation Types and Models' table, and look up the Device Type and Model found on the green screen. Input the number (1-15) found in the corresponding Value field.
• charset	
• codepage	
• workstationtype	
functionkeypatterns	Regular expressions used to recognize function keys. <functionkeypattern pattern="VAL" fulltextgroup="VAL" numbergroup="VAL" textgroup="VAL" />
• pattern	Any valid Regular Expression.
• fulltextgroup	Matches the entire text of the function key.
• numbergroup	The number of the group within fulltextgroup which matches the number of the function key (which will be the value submitted when the button is pressed).
• textgroup	The number of the group within fulltextgroup which matches the text of the function key.
autoformatpatterns	Regular expression use to recognize lines that should be subjected to autoformatting. <autoformatpattern pattern="VAL" />
• pattern	Default: [.]{0,1}(\.){2,}[ ]{0,2}:? Valid: Any valid Regular Expression
pollserver	Enable server polling (0/1). This increases network traffic significantly, but can be useful for programs that alter the screen or fields without reloading the page.
pollinterval	How frequently to poll the server (in seconds).
mousewheel	Support mouse wheel (table, none).
programtitle	Set this option to 1 to use the green screen program title as your browser page title. If your program titles aren't consistent enough for this to work well, set the value to 0 to make the browser page title 'Presto'.
errorhandling	Selected error handling style.
menulinks	Show menu links (Y/N).

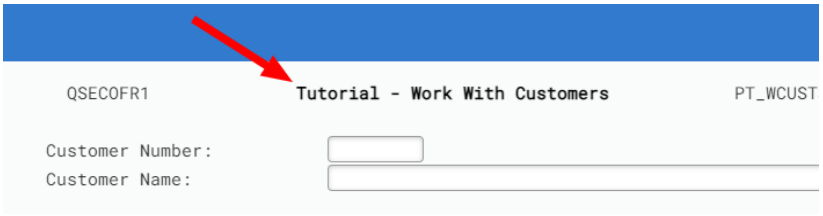
Config Value	Description
• version	Version of menu links. Default is 2.
• format	Indicates whether to transform the menu links into links or buttons.
• addclass	
• hidevalue	Hide the menu option numbers or literals.
• valuepattern	
• labelpattern	
• rtllabelpattern	
• targetfieldselector	Use this option to specify the target field for a menu converted into clickable links with the menu-links transformation. You normally don't need to change this value because the menu option is returned to the default input-capable field on the screen. However, if the screen has multiple input capable field, you will need to use this option to specify which field to return the menu option to.
detectsubfileoptions	Indicates whether the option to detect subfile options to create a flyout menu is enabled.
detectlastsubfileoption	Regular expression used to define criteria for subfile options to be converted to flyout menu options.
subfileoptiontrigger	Use this option to set the behaviour of the subfile flyouts. They can either be to set to display only when clicking on the row, or only on hovering over the input field. (clickonly/hoveronly)
functionkeyposition	Indicates whether function keys are displayed on the left, right or bottom of the screen.
disabletabprocessing	Set this option to 1 to disable Presto's tab ordering, and use the browser's settings.
CheckForBetterLayout	Detects which layout to use depending on the platform. Typically chooses between mobile or desktop layouts.
CustomPageReplaceSpans	Set this option to 1 to replace text constants with <screen> tags in Custom HTML.
findfunctionkeyinprotectedfields	Find function keys in protected fields. (Y/N).
disableoutputonly	Disables output only screens (Y/N).
reconcilewindow	Attempt to reconcile older methods of window detection (Y/N).
serverpositioning	Perform absolute positioning on the server (Y/N).
removeblankspans	This option is set on (value=1) by default, and is related to a fix in version 4.7 that allowed us to filter and ignore certain types of blank spans. The option is only available in case users of previous versions had tied any custom features to these blank spans.
ddswindowdetection	Allows you to control whether Presto detects non-DDS windows (windows that are drawn in manually, not using the proper keywords).

Config Value	Description
	Set this option to 0 if you do NOT want to detect windows based on a pattern, but only detect windows that are explicitly found in the DDS.
outputonlywait	The number of seconds to wait for a server response in output only programs.
sqldateformat	Allows you to change the SQL date format. See Knowledge Base Article 10485 for details.
ParametersVersion	Allows you to control whether you upgrade to the latest version of the software parm passing logic.
autodescription	This option corresponds to the Auto Detect Panel Description in your Global Settings.
checkserverresponsewaittime	This option corresponds to the Check for Server Response option in the Server Settings.
FunctionKeyContainerVersion	
OaUseDisplayFile	Set this option to zero if you do NOT want Presto to attempt to extract initial screen layouts from the display files of OA programs. Default value is 1.
OaHashFields	Adds the ability to exclude fields from hash creation.
FunctionKeyDetectionRange	Maps to Tools>Environment setting “Function Key Detection Rows (from bottom)”.
EnableStrPcCmd	See the section “Access PC Applications from Your Presto Site” on page 99 for an explanation of this option.
StrPcCmdPort	Port used by the StrPcCmd Server.
EnableDataScrubbing	This option corresponds to the environment-level Data Scrubbing option. The default value is 1.
DataScrubbingExpression	Set Presto to validate input data (ie.typing into a field). For example, if you wanted to validate input data for &, <, >, “, ‘ set the following: <DataScrubbingExpression>&|<|>|"|'</DataScrubbingExpression>
EnableDataValidation	Maps to the Tools>Settings Validate Data option.
DetectSubfilePattern	Presto incorporates complex algorithms to attempt to automatically detect tables, and where a table has been detected, determine if it’s also a subfile. Part of the process of recognizing subfiles is evaluating whether there are subfile options in the 2 lines immediately above the table. This value is the pattern Presto uses to make that determination.
Enable5250GuiCharacters	Toggle support for 'GUI characters' which are needed to support scroll bars in some cases.
WindowContainer	jQuery selector for the flattened window container.
ClearWindowContainer	Clear the content of defined window container. Values 1 or 0. Default is 0.

Config Value	Description
DisplayWindowAs	Choose if you want it to display it as a window or a panel. Default is window.
DetectTablesInWindows	Set this option to 0 if you do NOT want Presto to detect tables in windows. Default value is 1. Note: This option is only applied when the window's hash is built. For example, if you set this option to 0 and browse to a window that was already visited when this option was set to 1, then Presto will try to detect tables in that window.
IncludeWindowDimensionsInHash	Set this option to 0 if you do NOT want Presto to include the window dimensions in the hash. Default value is 1. Note: This option is only applied when the window's hash is built. It only affects windows that have not been visited yet.
TableFinderFilterMenuPattern	Default is 1.
TableFinderAllowTwoRowHeader Detection	Default is 1.
AdvanceCursor	Choose if you want to automatically advance the cursor to the next field when the field's length is reached. The default is to auto advance. Default value is 1.
GoogleAPIKey	This is a key required to use Google Maps. You can obtain this key with your Google account. Refer to the knowledge base article KB#15349 for instructions on getting this Google Map key and applying the key to your site.
OnInitFocusSelectFieldValue	Set this option to 1 to automatically select the content in the initially focused input field. The default value is 1.
DatepickerShowOn	This global option corresponds to the environment-level showOn option found on Date Picker fields. Choose if you want to have the datepicker appear automatically when the field receives focus ('focus'), appear only when a button is clicked ('button'), or when either event takes place ('both'). See the section on Datepicker Auto-Detection in Appendix B for more on this option.
DatepickerDateFormat	This global option corresponds to the environment-level dateFormat option found on datepicker fields. Choose the format for parsed and displayed dates.
ParametersVersion	The parameter passing is versioned to allow you to control whether you enable a change on upgrade. The default value is 2.
PasswordCapsDetect	Presto can indicate that caps lock is turned on when the user enters their password on the login screen. Set this value to 1 to enable this warning dialog. The default value is 1.
EnableDspfDetection	Set this option to enable detection of display-file changes. See the section on Document Dual Maintenance for more on this option.

Config Value	Description
Debuglevel	The detail level of the generated debug logs. The possible values are: critical, all, info, timing, error, none. The default is critical. The logging value will log all messages of that type and lower. 'timing' is used to record performance metrics without significantly affecting Presto's performance.
ValidateFoundWindow	Set this option to disable window validation. Default is 0. Note: This option is to disable window validation for certain edge cases where the validation prevents window detection.
UseStoredSubfileDetails	This option is the ability to enable and disable caching of detected subfile details. Default value is ON. Setting this option to ON improves performance. This option is to disable caching of subfile details for certain edge cases when, due to panel detection settings, different screens are matched as the same screen. If one of the matching screens has a table, it will be displayed on both screens.
SubfileRequired	Setting this option to 1 allows the SubfilePaging module to convert page-up and page-down text to buttons even when there is no subfile present on the screen. Default is 0.
AllowContextMenu	Setting this option to 1 in the JavaScriptOptions module enables the browser right-click menu. Default is 0.
MaxColumnDistance	Set the maximum number of spaces between the MenuLinks numbered option and the title of the menu option in a row. When detecting MenuLinks, Presto evaluates the number of spaces between the numbered option and the title of each menu option. MenuLinks are only detected if the distance between the numbered option and the title of each menu option is less than the MaxColumnDistance. Default is 4. Note that increasing this value could result in an increased number of incorrect detections of MenuLinks.

PageHeader	Setting this option to 0 in the PageHeader module disables the Page Header. Default is 1 (enabled). Add the following to the <module></module> section to disable the header:
	`<module name="PageHeader" enabled="0">`
	`</module>`
	Here is an example of a screen with a disabled Page Header:

Config Value	Description
	
AdjustLastRow	Setting this option to 0 in the JavaScript module turns off the feature to move elements on row 24 up when function keys are moved. After moving function keys to the left or right sidebar or the Navigation Buttons module, the screen contains empty rows 22 and 23. This option adjusts row 24, moving it up, to remove the empty rows. Add the following to the `<javascriptoptions></javascriptoptions>` section to disable the adjustment: `<AdjustLastRow>0</AdjustLastRow>` Default is 1 (enabled). Note that this option can be set for individual screens"Adjust Last Row" on page 139 of Chapter 2.
TableTouchActionVersion	Set the action/swipe behavior on tables for mobile devices. Default is 1. Setting the option to 1 uses the same behavior that has existed since Presto 8.0. When on mobile devices, swiping on tables will scroll the entire screen. You must first scroll to the top or bottom of the screen then keep scrolling to page the table. Setting the option to 2 will mean swiping on tables will not scroll the screen, and Presto will only page the table. This behavior only applies to mobile devices using non-responsive skins and responsive skins with screen widths > 880px. For responsive skins with screen widths <= 880px, behavior defaults to option 1 because the table will reflow at this width and take up the entire screen.
MaxDqCount	Setting the option to 0 or 1 mean the same, which is 1 data queue PR_CONVQ Setting the option to 2 or greater value will tell Presto to use a data queue with the following names PR_CONVQ00 - PR_CONVQNN See "Multiple Data Queues" on page 418 of Chapter 2.
changemanagementoptions	
• scrubdata	When exporting pages, setting this option to 1 will remove all fields that are not utilized during the import process and resequence IDs for customizations.
• backupdir	The location to save backups of customizations when importing screens

Config Value	Description
	and replacing existing pages. If left empty, the system automatically uses the path of the files being imported as the backup location. In cases where a relative path is provided, the system utilizes the IFS path of the instance as a base for creating backups.

Using the Presto Designer

Once you have defined your connection to the server you are ready to start using the Presto Designer to customize the appearance of your programs.

Here is an illustration of the Main Menu on our IBM i, after signing on from the Sign On screen shown in Figure 2-1:

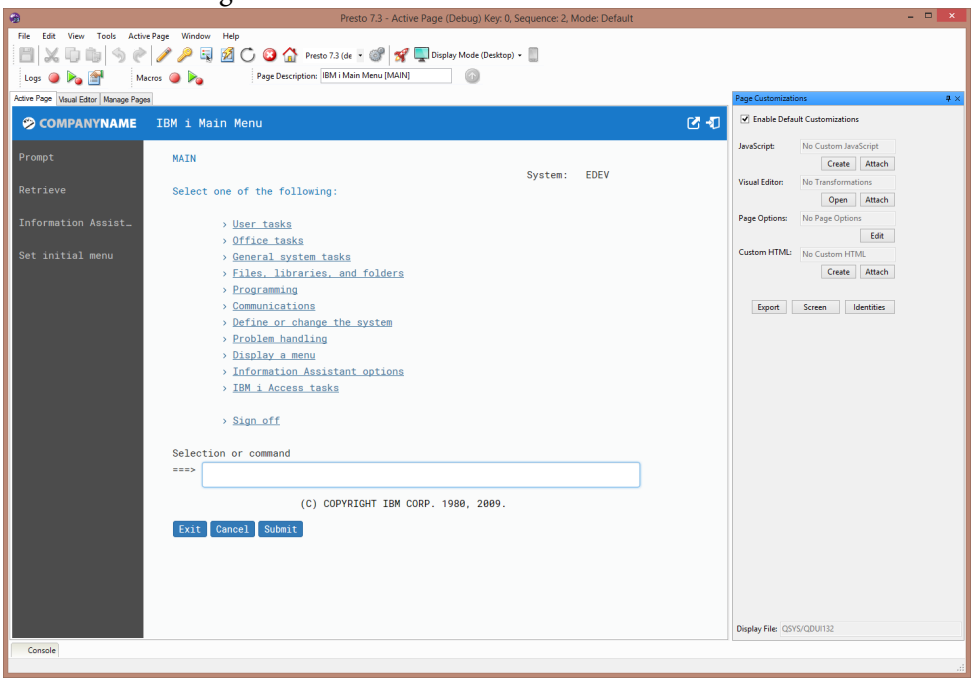


Figure 2-17. The Main Menu.

This is presented using the default settings in the Pacific skin.

The Visual Editor tab (at the top left) is described later in this guide. The main Presto Designer options and features and other tabs are discussed below.

Editing the Page Description

As you navigate your applications Presto is creating an internal database of your pages so it can recognize them later to correctly apply your custom changes. While Presto is using a combination of several different page structure and content elements to build its [internal] page recognition criteria, your own link to that information is via the Page Description.

The page description is by default extracted from the area of your screens typically occupied by a page title and displayed in the toolbar. Here is an illustration from the tutorial application:

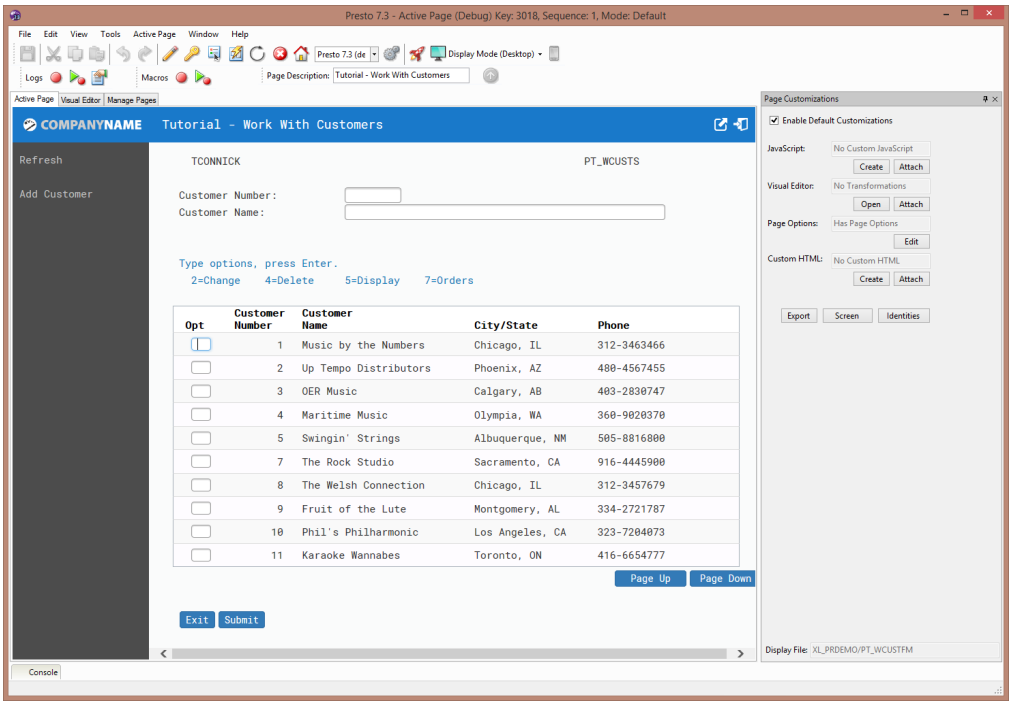


Figure 2-18. Reviewing and editing the page description.

In this example the default page description is “Tutorial - Work with Customer Orders”, which Presto has extracted from the title of the page. As you navigate your own applications the quality of the automatic page descriptions may vary depending on your page layouts and the quality of the descriptions in your own applications. If your page descriptions seem vague, repetitive or generic, change them. Click the button to the right of the text box



It is important to monitor the page descriptions as you work with your applications to make sure they are both meaningful and unique. You can change the default location in which Presto looks for the title from your site settings (Tools>Settings>Detection Settings).

to save your change.

Testing Pages in a Browser

The Presto Designer includes a handy shortcut for opening your current environment in a browser. You can use this when you’ve made some changes to a page that you want to test.

Launch your browser from this button in the Presto toolbar:

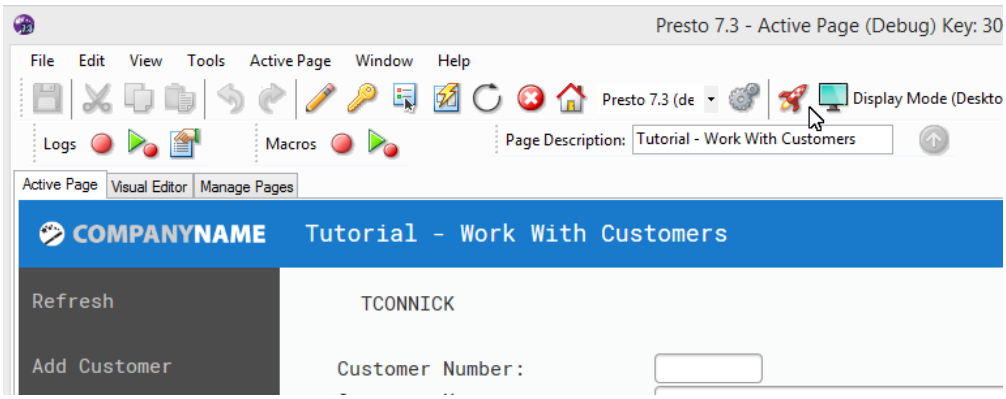


Figure 2-19. Launching the current environment in a browser.

When you click the button Presto will pick up the URL from your current environment's connection settings. Sign in and navigate to the page you need to check.

Using the Find Window

When you open custom HTML or JavaScript files, or edit any other kind of file in Presto, you can use the Find window (Ctrl+F) to look for text strings and other content. Here is an illustration of the supported options:

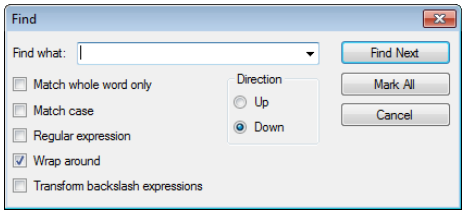


Figure 2-20. The Presto Find window.

Most of the options are self-explanatory, but here are brief descriptions of those which may not be so clear.

Mark All

Use this option to have Presto flag every line that includes a span:

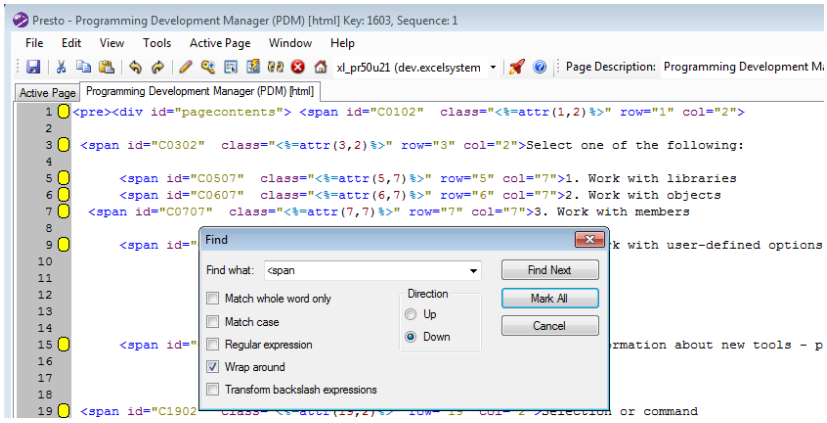


Figure 2-21. Marking every line that includes a matching string.

Regular Expression

Check this flag to have the Find window interpret your search string as a regular expression, and look for any matching patterns. For example, checking this box and searching for `<%=field (\d+,\d+,\d+)%>` would find every instance of a field in a custom HTML page (such as `<%=field (20,7,153)%>`).

Transform Backslash Expressions

Check this option to search for Backslash Expressions. The most common are \r, \n, \t. For example, if you wish to remove all empty lines you could search for \r\n\r\n and replace that with "".

Changing the Layout of the Presto Designer

You can change the default layout of the main sections in the Presto Designer by dragging & dropping them in a different location, or tearing them away to function as standalone windows. To do this, click and drag the header section of the window off the main designer window, as shown here:

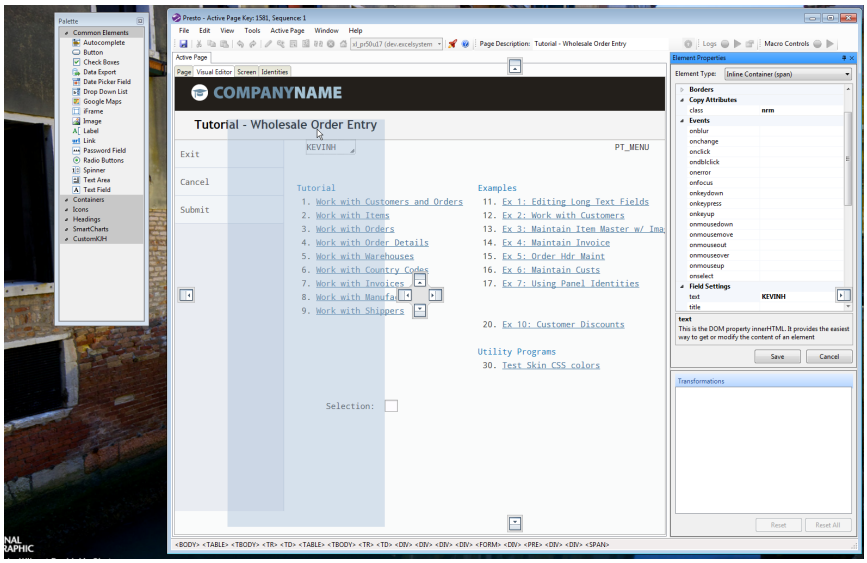


Figure 2-22. Dragging the side options out of the main designer window.

In this illustration the Palette tab is already outside the main window, and we are in the process of dragging the Element Properties window outside as well. To move the window back drag it back over the main window and insert it into the location indicated by the tabs that appear around the edges of the main window.

If you don't have any experience with tear-away tabs and windows feel free to play around, and if you make a complete schmozzle of the Presto Designer call Technical Support and we'll help you put the pieces back together!

Hiding and Restoring Designer Sections

You can hide or remove sections of the IDE using the X in the top right corner.

If you hide a window element (the Page Customizations, Element Properties, Page Identities or the Green Screen view), you can restore it via the View menu.

The Active Page Tab

In this illustration the main window is displaying the Active Page tab:

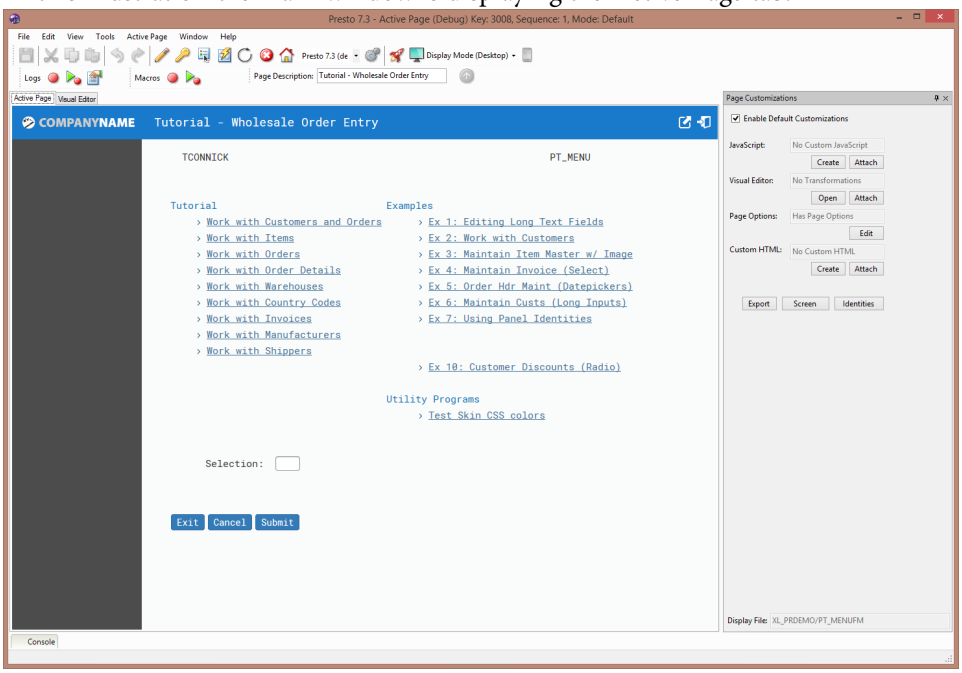


Figure 2-23. The Page tab in the Presto Designer.

From here you can work with a preview of what your users will see when they use Presto to run their programs in a browser.

In the Active Page area you are actually running your programs - you can work from here just as you would from a standard 5250 session. You can call your programs or menus to see how your own applications run in the default Presto environment.



Keep in mind that when you work with your applications via the Presto Designer they will run significantly slower than they will in a browser (which is the way your end users will access their applications). The designer does lots of extra processing as you navigate through your pages and performance during your design sessions is unrelated to how fast your applications will load in a browser.

But don't forget that when you test, you are changing real data!

The Screen View

Use the Screen button to check the appearance of your actual IBM i program:

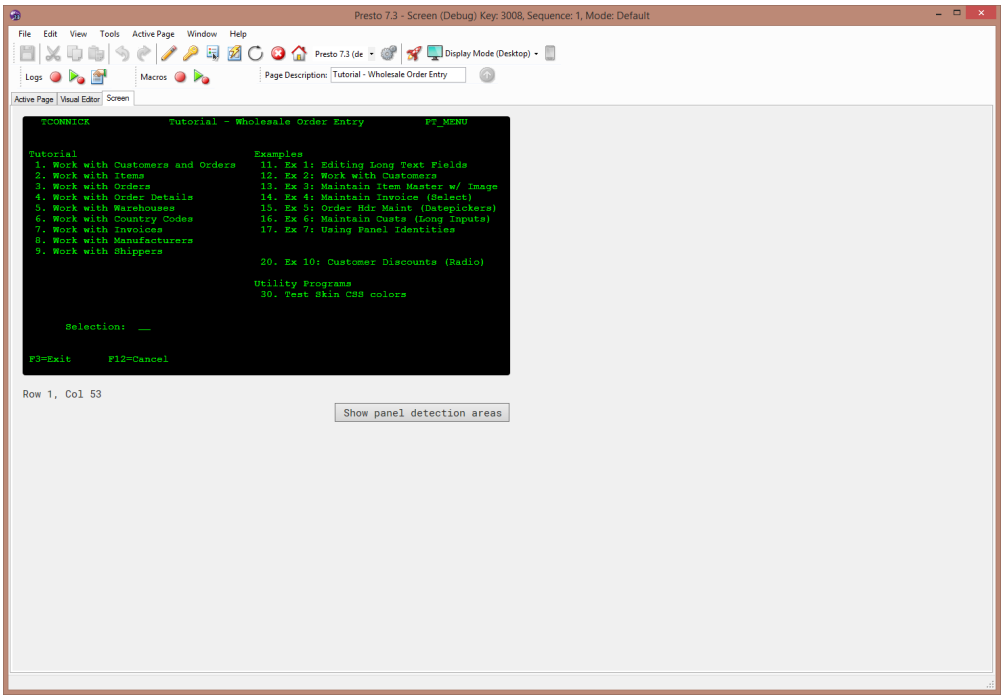


Figure 2-24. The appearance of the panel on the IBM i.

This view provides you with a touchstone as you customize the appearance of the page to be presented in the browser. You'll also use this view when you create or edit HTML tables on your pages.

This is a current image of the screen but not an active display. You can't use your program here, though you can toggle back to Browse at any time.

From here you can also press the 'Show panel detection area' button to highlight the sections of the page that Presto is looking at to determine the page's identity.

Working with Page and Global Identities

Access your Page and Global Identities from the button on the right of the Active Page:

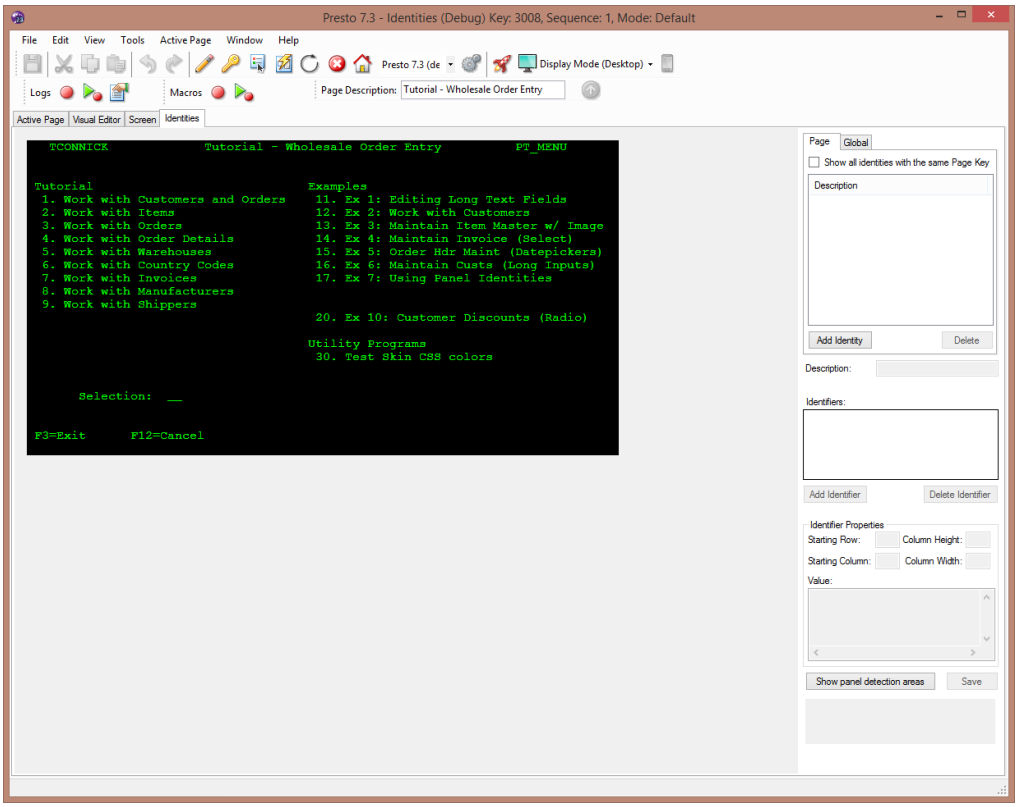


Figure 2-25. The Identities tab.

To understand this feature it's useful to step back a couple of paces to review what Presto is doing as it renders your applications.

As you run Presto...

As you use the Presto Designer to navigate through your programs the software automatically identifies and records unique characteristics of every screen. It does this so it can reliably recognize them later and correctly apply any custom changes you add.

Panels are identified using:

1. The page areas specified in your screen detection settings (see "Detection Settings" on page 40).
2. The fields on the page (optionally).

Armed with these two pieces of information, in most circumstances Presto can uniquely identify your screens, distinguishing one from another.

However, it is possible to have different screens that are identical in Presto's eyes. This typically happens when the distinguishing features lie outside the auto detection areas that work for the majority of your other programs. The visible symptom of this issue is that Presto applies a custom feature to the wrong screen. You can address this by adding 'identities' to your panels, instructing Presto to add more pieces of data to its screen identification details, to help distinguish between otherwise identical screens.

Some environments can present unique challenges for consistently identifying pages automatically, and in some situations, you may experience collisions between page level

identities. In these circumstances you can also use Global Identities to identify some of your pages.

In the section “Auto Detection Settings and Identities” on page 114 of the tutorial you can find step by step instructions for creating panel identities, and the section “Using Panel Identities to ID Screen Variations” on page 122 of the same chapter includes a detailed, illustrated discussion of when and how to use global identities.

Device Preview Dropdown

Use the device-preview dropdown to preview your Presto site in common mobile-screen resolutions. The option is available in the toolbar, next to the Launch button.

You can open the device-preview options from the toolbar dropdown:

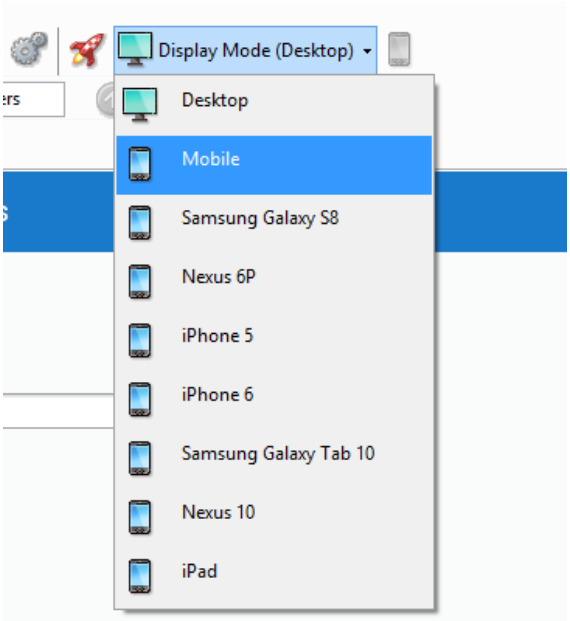


Figure 2-1. Accessing the device-preview dropdown

The currently supported resolutions are as follows:

Device	Width (px)	Height (px)
Samsung Galaxy S8	360	740
Nexus 6P	412	732
iPhone 5	320	568
iPhone 6	375	667
Samsung Galaxy Tab 10	800	1280
Nexus 10	800	1280
iPad	768	1024

Working with Custom Files

Presto allows you to add files to your /userresources/ IFS folder via the designer's Tools>User Resources>Add files option:

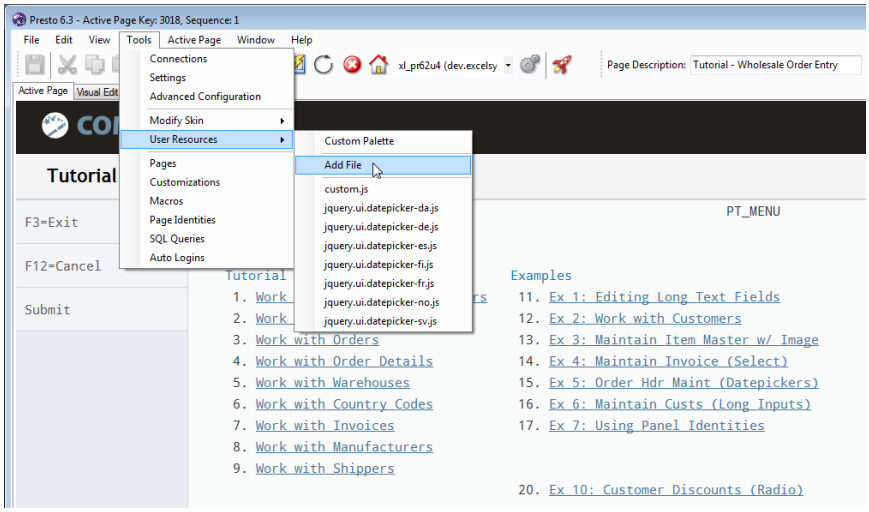


Figure 2-2. Selecting the option to add a file to your custom resources.

When you select the option above Presto presents a standard 'Open' dialog, where you can navigate to and select the file to upload.

Once you upload the file it will be automatically added to the list shown above (below the Add File option). From here you can open and edit the file, and once it is open, you can also right-click the file's tab in the Designer to save, close or delete it.

Custom Files Shipped with Presto

Notice that in the list above there are a number of existing files. These include custom.js and a small set of datepicker localization files. You can edit these using the Presto Designer.

Support for Saving Files Locally

When you open a file of any kind in the Presto Designer you can right-click the file tab and select Export... option to save a local copy:

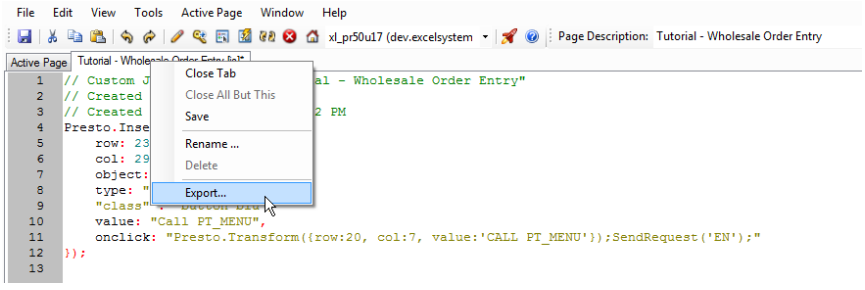


Figure 2-3. Preparing to export and temporarily save a file to our desk top.

This feature is useful for keeping temporary versions on your PC while you work on changes, or if you need to attach the file to an email to Technical Support.

Managing Pages and Customizations

As you work with your applications in Presto the software is keeping track of all the pages you visit and any custom changes you make to them. From the Manage Pages window you can work with this list.

To manage these components select the Pages option from the Tools menu. Here is an illustration of the window presented:

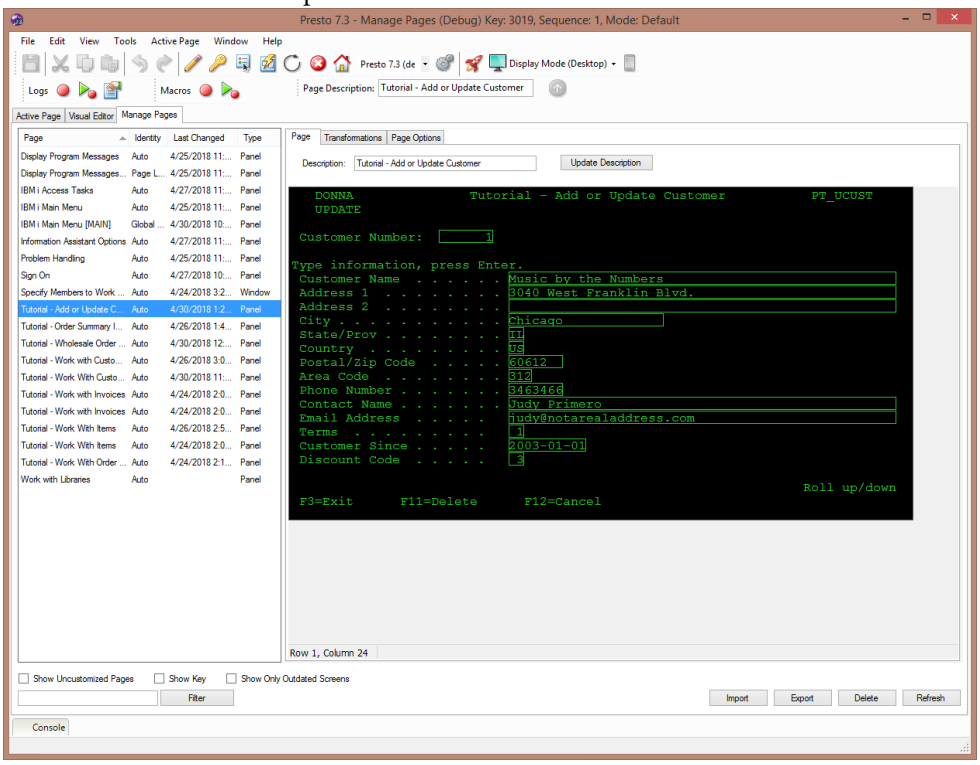


Figure 2-4. Presto's Manage Pages interface.

From here you can work with all your pages (customized or not) as well as with the individual customizations on those pages. The sections below describe the ways in which you will use these screens.

Filtering the Page List

Use the Filter option in the bottom left of this window to limit the list to a particular set of pages.

The Pages Tab

The default view, showing above, lists all the pages you have customized. When you click on an entry in the list the tabs above the window on the right allow you to review and work with the HTML, Page Properties, JavaScript and Transformations attached to the page.

Deleting Page Entries

You can delete page entries to clear the internal record for that page and any page option changes. Deleting a page also deletes the links between the page and any customizations or

custom HTML. You can use this as a way to 'reset' all the customizations for a page to start working on it again from scratch.



When you delete a page, the actual visual editor customizations and custom HTML changes are NOT deleted. Presto allows you to attach custom HTML and visual editor changes to multiple screens, and you may be using them elsewhere, or intend to re-use them later on the same page.

Show Uncustomized Pages Option

Presto doesn't just identify pages when you attach custom features to them - it automatically creates a record of every page you visit with the design tool.

To review all these records check the 'Show Uncustomized Pages' option on the bottom left of the Manage Pages window.

This feature can be a useful troubleshooting tool, in that deleting a page from Presto's internal files forces the software to look at that page with fresh eyes the next time you visit it.



As an example, this can be relevant in a situation where Presto didn't initially recognize a subfile as a list and you want to reset the software's records for that screen.

To explain: Presto identifies subfiles as such based on patterns on the screen. If the first time you visited a subfile program the subfile list portion of the screen was empty, or the subfile had a single record with oddly placed column headings, it's possible that Presto won't recognize the subfile portion of the screen as a repeating area that should be rendered in a table. If you then revisit the screen where multiple records are present (and there's a pattern to their arrangement on the screen) Presto may initially record it as a different page, and not apply the custom features you've created.

Showing Page Keys

Another useful troubleshooting key is the option to show the internal page keys. This can help you understand page collisions, where customizations are unexpectedly applied (or not) to a page. This is discussed in more detail elsewhere, but if you see duplicate keys for two page records, it would tell you that Presto internally believes them to be the same page. If you see multiple values for what you think is a single page, this means that Presto has seen a difference between the two pages.

Show Only Outdated Screens

Working with Customizations

You can also work with customizations, and from there see the pages to which they are linked. Here is an illustration of an entry accessed from that tab:

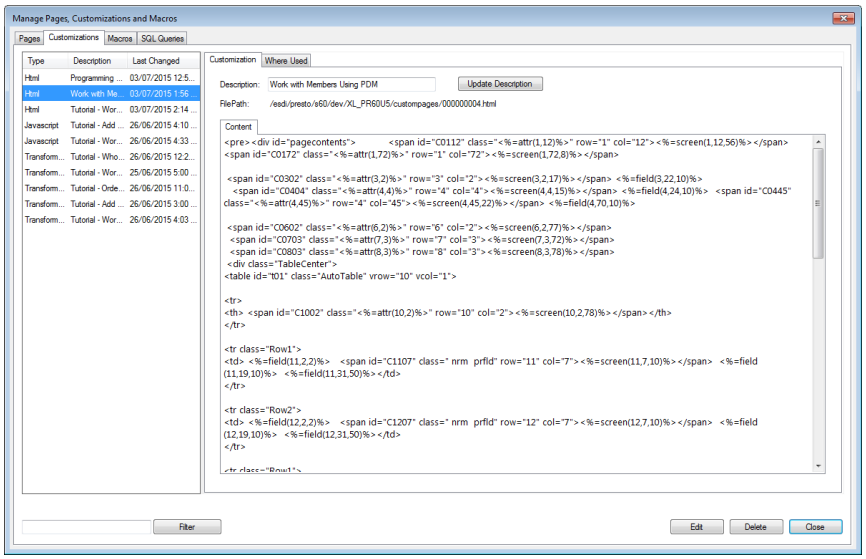


Figure 2-5. Displaying the customizations created in your environment.

The list on the left includes entries for every custom HTML and JavaScript file you create, as well as entries for every set of transformations you attach to a page.

Description

From here you can correct or improve your customization description.

Change the text and click the 'Update Description' button.

File Path

This is the actual IFS location of the transformation file.

Content

In this tab you can review the customization details.

Where Used Tab

Click this tab to see a list of the pages where this transformation is used.

Attaching Custom HTML, JavaScript or Transformations to a Page

While working with Presto you may occasionally detach customizations from their original screens so you can verify the screen's default (uncustomized) behavior, or to experiment with a different type of change. In some situations you may also want to attach the same customization to multiple screens in your application.

To manually attach a customization file to a page click one of the Attach buttons available in the Page Customizations area of the Presto Designer:

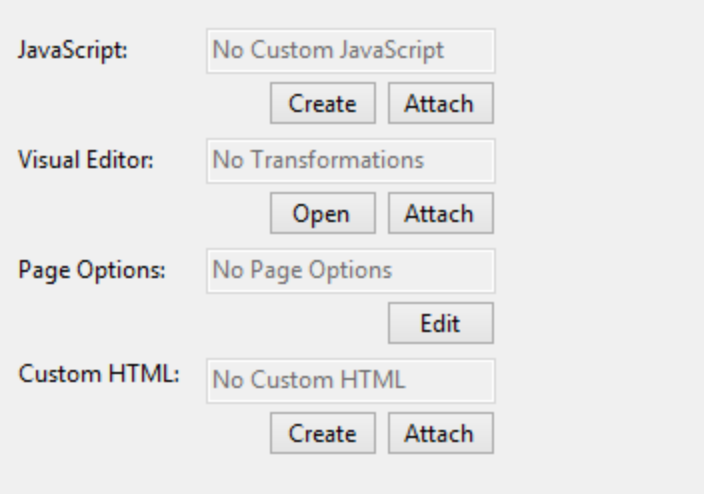


Figure 2-6. Options to attach customizations to your pages.

When you click any of these buttons Presto presents a window similar to the Manage Pages, but which shows only a list of customizations of that particular type (e.g. HTML, JavaScript or Transformations).

From the window presented you can preview the code in your customization as well as update their descriptions.

Creating and Working with SQL Queries

Presto allows you to create custom SQL queries which you can use in dynamic drop down lists and radio buttons, data exporting features, SmartChart transformations and autocomplete fields (among other features). You can use SQL queries in both the visual editor and custom HTML.

To work with your SQL queries select Tools>SQL Queries to open this window:

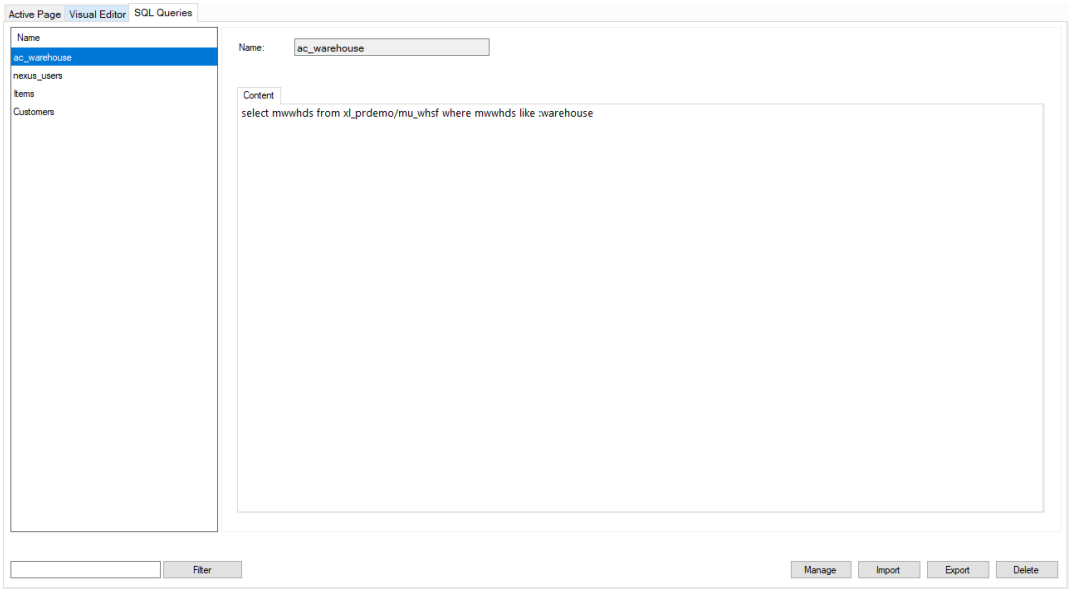


Figure 2-7. The window for working with SQL Queries.

To manage your SQL queries click on the Manage button:

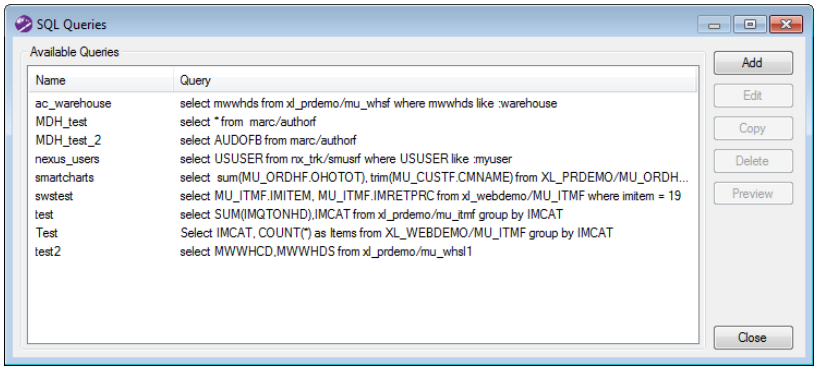


Figure 2-8. The window for working with SQL Queries.

Click the Add button to create a new query or double-click an entry to edit it.

This is the window for creating and working with individual queries:

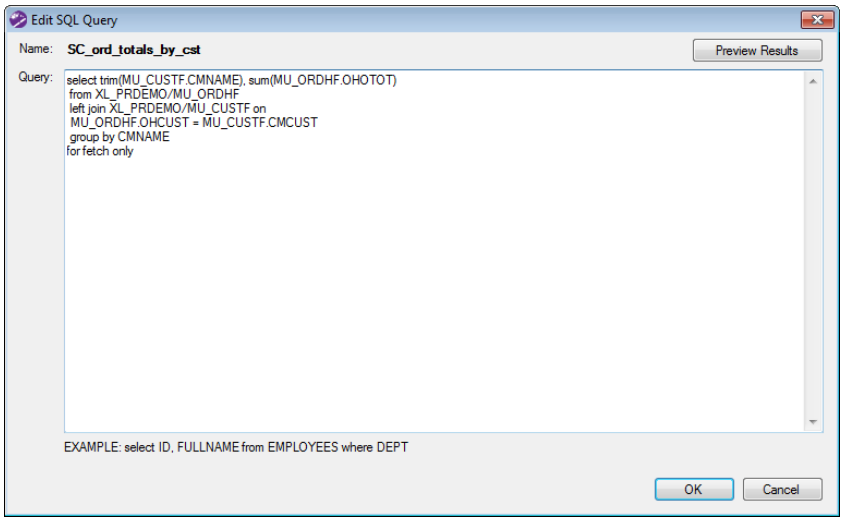


Figure 2-9. Creating or working with a SQL Query.

Query Name

The query name must be unique and cannot contain spaces.

As your list of queries may grow quickly we recommend that you establish a standard naming convention that includes key information, such as the file, the displayed value and the value returned (where relevant).

Query

Enter a valid query, using this basic structure:

```
Select Returned_Value, Displayed_Value from Library/File
```



Note that the library name is optional, as file access is by default controlled by user's library lists.

There are a couple of exceptions to this: SQL invoked on the Sign On screen needs to have hard-coded library names, and WebFacing is not supported with this feature.

Queries you intend to use with autocompletes only need a single field value:

```
Select Returned_Value from Library/File
```

You can also use parameters:

```
Select Returned_Value, Displayed_Value from Library/File where Field = :Parameter
```

You can then associate the parameter with a value returned from the screen.

Parameter names are limited to using the characters A-Z, 0-9 and the underscore (_). Special characters such as @, $ or & are not permissible.



Note that quotes are not needed in Presto queries or around parameters even if the field is alpha.

There are some additional nuances available in how you define your SQL queries for particular element types. These are documented in the following sections of , The Presto Visual Editor:

- “Autocomplete” on page 201 (includes an extensive set of examples).
- “Drop Down List” on page 226.
- “iFrame” on page 240.
- “Radio Buttons Transformation” on page 246.
- “Signature Pad” on page 250.

There are also examples later in this guide of using queries with custom scripts and custom HTML.

Preview Query Results

From the main query list, as well as the query maintenance window, you can preview your query results. When you click the Preview Results button, Presto displays up to 200 records in a window similar to this one:

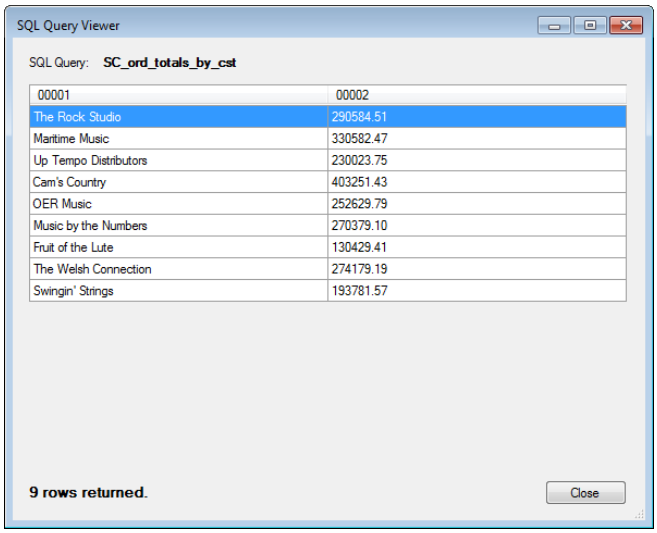


Figure 2-10. Previewing query results.

Use this accessible tool to quickly verify whether your query will return the expected results for your users.

You can also preview the results when your SQL statement contains parameters. Enter the values of the query parameters and click Run Query to see the expected results:

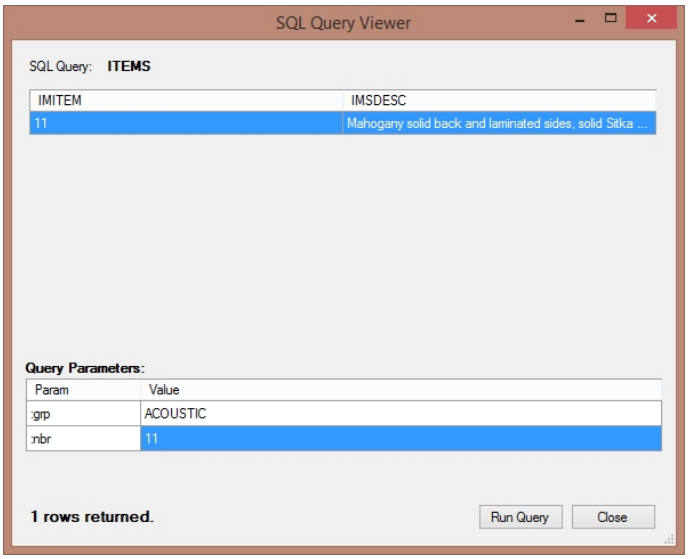


Figure 2-11. Previewing results for a query with SQL parameters.

General Considerations for Working with SQL Queries

Here are some other general tips and considerations when using queries:

- If you need to check what exactly a query is returning you can use the keyboard combination CTRL + Shift + ~ in your browser window to see the full HTML generated at run time. For instance, you can use this technique to review all the entries created in a drop down box.
- Parameter names must begin with an alpha value.
- Trim numeric return values if needed.

Test your queries thoroughly using STRSQL (on the IBM i) or an equivalent tool. To test queries with parameters assign a static value for your parameter.



A user with access to the Presto Designer can execute any SQL statement which they would normally be authorized to. A user of the browser interface can invoke any defined SQL statement, and it will run under their user profile.

The SQL statements themselves are not transferred from or to the browser client, and thus cannot be 'tinkered' with. SQL injection protection is supplied by the SQL server itself through the use of query parms.

Using SQL Queries in Custom HTML

Here is an example of an SQL query defined in the Presto Designer:

- **Query Name.** InventCatGroup
- **Query.** select ICGROUP, ICGROUP from xl_prdemo/mu_itmcf where ICINCA = :wfinvcat

Here is how you can use this query in a transformation in custom HTML to replace a 'Category Group' field on a screen with a dynamic drop down list:

```
<script type="text/javascript">
Presto.Transform({
row: 8,
col: 31,
to_object: "select",
sql_options:
{
   name: "InventCatGroup",
        params:
        {
        wfinvcat:"<%=screen(7,31,10)%>"
}
}
});
</script>
```

These are the key elements:

- **row/col.** The row/col must match the location of an existing field on the page. In this case, the field on the screen is the Item Category.
- **to_object.** The to_object type of "select" generates a drop down box.
- **sql_options.** The sql_options need to reference the query name, and if required, the parameter.
- **params.** You can specify a screen value (implicitly a dynamic value) as a parameter. In the example above, the parameter value is retrieved from the screen location 7,31,10 (the 10 positions starting at row 7, column 10). As you work with various records the SQL query grabs the parameter from the screen.



If you are using a screen value as a parameter, you need to make sure that you won't have records for which the specified value will be blank, or your page will generate errors.

When you save this change, Presto displays a dynamic drop down in place of the field.

Support for Dynamic Library and File Names in SQL

You can use custom defined variables in SQL queries. Here is an example of a query that loads a warehouse location:

```
select MWWHCD,MWWHDS from {lib}/mu_whsl1
```

Note the replacement value {lib}, which allows your pages to dynamically assign a library name before executing the query, as shown here:

```
Presto.ExecuteSql({
    format: "json",
    dataType: "json",
    name: "WH_Location",
    placeholders:
    {
```

```
            "lib" : "xl_prdemo"
    },
            success: function(data, textStatus, XMLHttpRequest)
    {
             //do something
    }
});
```

Example: Setting a Field Name

Here is another example, with this SQL query named 'placeholder':

```
select cmcust, {field} from XL_PRDEMO/MU_CUSTF
```

This is the corresponding JavaScript:

```
self.ChooseField = function(sField)
{
        Presto.ExecuteSql(
        {
                name: 'placeholder',
                format: "json_array",
                dataType: "json",
                placeholders:
                {
                        field : sField,
                },
                success: function(data, textStatus, XMLHttpRequest)
                {
                        Presto.Transform(
                                {
                                        id: 'f0',
                                        to_object: 'select',
                                        options: data
                                });
                        }
        });
}
```

And in the Visual editor you can do this:

```
onclick: var sField = jQuery('input[name=xlg0002]:checked').val(); ChooseField(sField);
```

Call Technical Support if you have any questions about this!

Using jQuery Selectors for SQL Parameters

Presto supports the ability to use screen values in your SQL queries, which allows you to make your queries dynamic, reflecting the content of particular screens. This approach is useful for situations where you want your query to reflect a value on a screen, and when that value will not change while you're at that screen. However, this approach has some limitations with incorporating values that change based on user actions (e.g. selecting a value from a drop down list, a radio button or a check box).

You can use jQuery selectors to address this need, allowing you to do things like generate autocomplete values within a range limited by a previous selection in a drop down list.

Syntax

The most popular selectors are ID and Class.

- selector:#ID - by ID
- selector:.className - by class

Any jQuery selector is valid

Technical Notes

All Presto HTML elements have a unique ID which is listed in the properties list in the visual editor. The one exception to this is elements you add manually without an ID on a custom HTML page.

This feature selects the Value attribute. It won't work for span or div since those elements use Text attribute.

Examples

Here is a sample of the steps to follow to use this feature:

1. Create a dropdown list. Add the following list values: 22,4. Write down the element's ID.
2. Create an autocomplete element and associate it with this query: select distinct MMMANA from xl_prdemo/mu_manf where lower(MMMANA) like :manf and MMMFG=:parm2
3. Set the parm2 value to: selector:#dropDownListID
4. Set the autocomplete Term to manf.
5. Save and return to the Active Page.

To test this, enter 'r' in the autocomplete field. Now change the dropdown value and try 'r' again.

Another example using the tutorial files is the City autocomplete, using the selected state as part of the query. With this feature, if the user changed the state, the new state field value would get used.

This would be the SQL selected:

```
select trim(CTCITY) from XL_PRDEMO/MU_CITYF where CTSTAT = :state and CTCITY like :city
order by CTCITY
```

Now the :state can be the CURRENT value of the state field, not some value that was presented when the page was first loaded.

Recording and Playing Macros

Presto supports the ability to record and play back macros. Because macros are played on the server and only return control to the browser upon completion, they allow much faster navigation between screens - your users can drill deep into an application with a single button click.

You can attach macros to screen elements for your users.

Recording a Macro in the Presto Designer

The macro controls in the Presto Designer are simple and easy to use. This is the initial appearance of the controls in your toolbar:

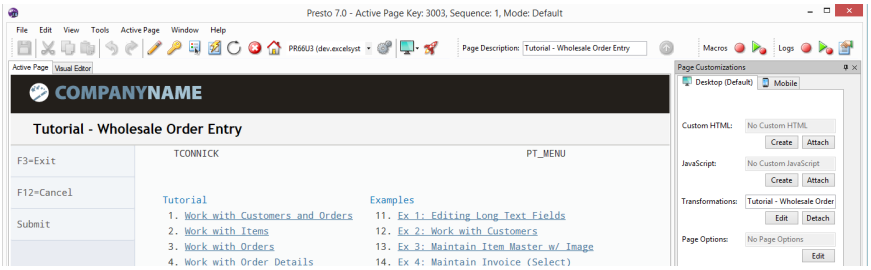


Figure 2-12. The macro toolbar.

The Presto Designer toolbar is configureable and the exact location of the buttons may vary.

Starting and Stopping a Macro Recording

Before beginning your macro recording navigate to your starting point (be it a menu or application). Then click the red button on the left.

Once the recording has started the button will turn into a blue square. After you have completed the steps you want to repeat, click the button to stop the recording.

Saving or Discarding Recordings

When you stop a recording you can choose to either save or discard it:

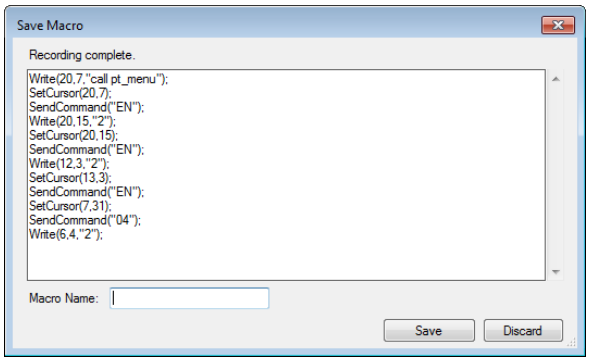


Figure 2-13. Completing a macro recording, and preparing to save or discard.

Type a macro name and click the Save button to add this macro to your library, or click Discard to delete it.

Replaying Macros in the Presto Designer

You can use macros in the design tool as shortcuts to jump to a particular spot in your application, or you can attach to screen elements for your users.

To play a macro in the design tool click the green arrow icon in the macro toolbar. When you do this Presto prompts you to select a macro:

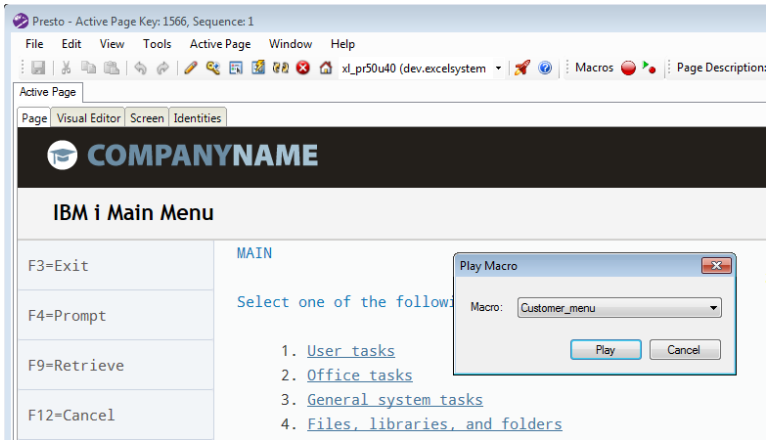


Figure 2-14. Selecting a macro to play back in the designer.

Select a macro from the list and click the Play button (under the dropdown shown above).

Playing a Macro from a Page

You can attach macros to screen elements to make them available to your users. In this illustration we have attached an existing macro to a new button on a page:

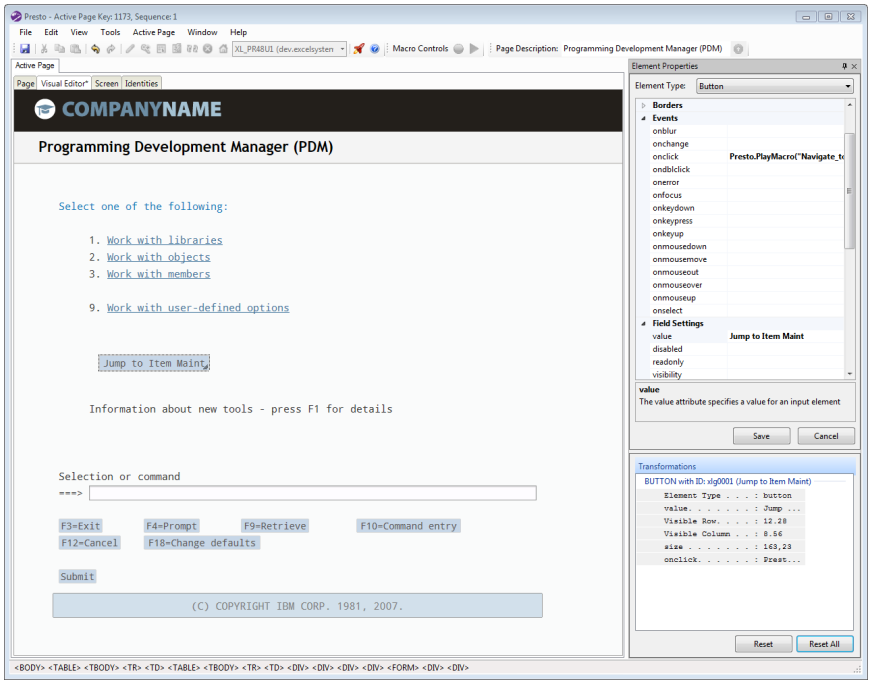


Figure 2-15. Adding a new button that executes a macro.

Here is the full line of script in the Event Editor:

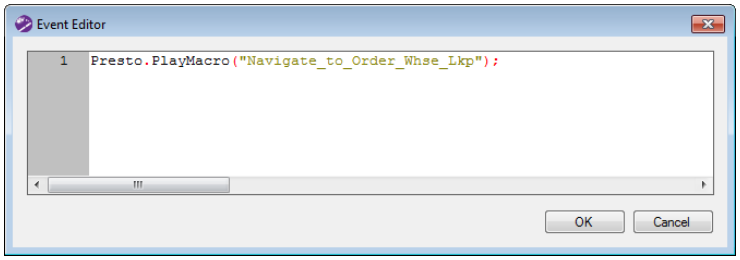


Figure 2-16. The script in the event editor.

This is the line we added:

```
Presto.PlayMacro("Navigate_to_Order_Whse_Lkp");
```

Insert the name of the macro you recorded and save your code.

Running Macros Sequentially

If you have various nesting levels in your menus and want to add macros to drill to varying degrees of depth, you can record macros as building blocks and run them sequentially.

This is an onclick event for a button that traverses three layers of menus:

```
Presto.PlayMacro("01_Call_Menu");
Presto.PlayMacro("02_Select_Opt_2");
Presto.PlayMacro("03_Select_Menu_4");
```

Some of your veteran users may have various macros recorded to simplify navigation through their applications on the green screen, and the ability to string together smaller scripts into longer ones may be useful!

Managing and Editing Macros

You can work with your recorded macros from the Tools>Macros menu in the Presto Designer:

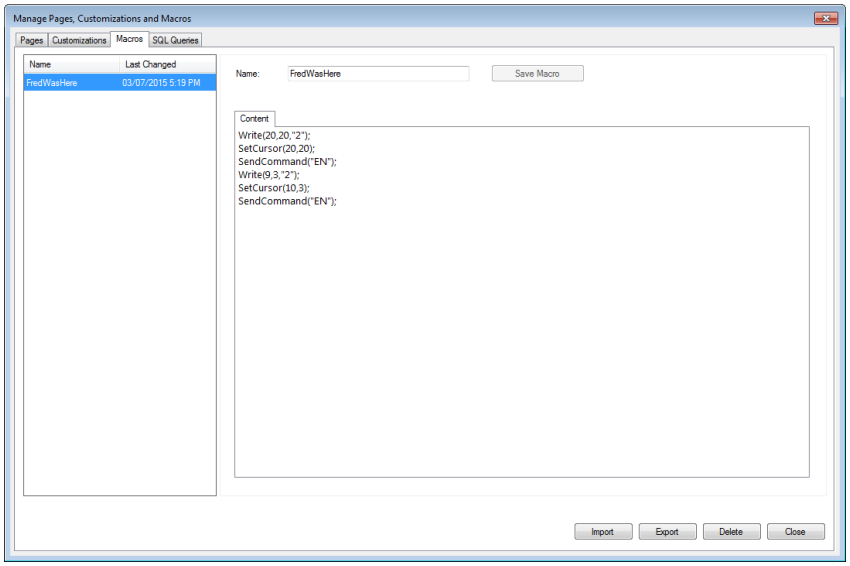


Figure 2-17. Working with macros.

From this interface you can change your macro names or scripts, as well as delete macros that you don't need.

In addition, you can edit existing macros to extend or crop them, or change sections.

Here are some examples of macro commands:

Command	Explanation
Write(20,7,"call pt_menu");	Writes text to the screen.Requires X and Y coordinates, and a text string.
SetCursor(20,7);	Sets the cursor to a particular position. Requires X and Y coordinates.
SendCommand("EN");	Submits a command key.
Expect(2, 25, 'MAIN');	Assert that the text starting at position (row, col) on the current screen matches the provided value. If it does not, the macro will be terminated and an error returned. **Note: This should only be used if calling a macro from the Presto API.**
GetScreen();	If GetScreen is called in a Macro, the current screen at that time will be returned to the Callback method instead of the last screen. There MUST be a SendCommand at some point after using GetScreen. If it is missing, the Macro may hang the session. See "Presto JavaScript API" on page 448 of Appendix 2

Most macros are a combination of these three simple navigation commands.

In addition to passing the name of a saved macro, you can also pass a JSON list of commands, as the following example illustrates:

```
Presto.PlayMacro({
macro: 'Write(20,7,"al xl_prdemo");
        SetCursor(20,19);
        SendCommand("EN");
        Write(20,7,"call pt_menu");
        SetCursor(20,19);
        SendCommand("EN");'
});
```

Creating and Working with Auto Logins

Auto logins allow you to configure a way for authorized users to go directly to a program/menu without an additional login via Presto. This feature is primarily intended to be used with Nexus, Fresche's web portal, although you can also tie it into your own portal software. If you use auto logins with Nexus, you can add URLs to the Nexus ECM (Enterprise Content Manager) and have users logged in automatically using their Nexus IBM i user profile information.



We also have customers who have implemented Single Sign-On (SSO) solutions with Presto and Kerberos. Knowledge base article 10295 at our website describes how to set this up.

Technical Requirements

There are a number of technical requirements that are critical to successfully implementing auto logins.

Nexus Profiles

To use the auto login feature your Nexus profiles must be connected to IBM i User Profiles.

Nexus Program Parameters

Auto logins currently allow you to call a program, but not pass any parms.

User Profile Must Have "Limit Capabilities" Option Set to *NO

In order for auto logins to work, your user profiles must have the "Limit Capabilities" option set to *NO. If it is set to *YES, a number of constraints are imposed on IBM i user profiles which will not allow the auto login feature to work.

Apache HTTP Configuration Requirements

When you use auto logins, the login information is passed by cookie. One of the constraints when working with cookies is that you can only pass the login information to servers of the same domain.

For example, suppose you are running Presto with this URL: `ibmi.example.com/presto/presto`, and you have your own portal system at `www.your_domain.com`, and you want to log in to Presto from your portal. In this case you can't set the autologin cookie information for `ibmi.example.com` while on the `www.your_domain.com` web page directly; you will need some other way to set that information.

A possible workaround is to go to an intermediate page before going to Presto, but that's not always desired or a possibility.

The best approach is to merge the Apache HTTP configurations for Presto and Nexus.



Another reason we recommend merging the Nexus and Presto Apache server instances is that Presto will try to resize the portal to fit the page into which it is being loaded. This requires calling a function in the Nexus page.

For security reasons, browsers only allow this if the port and domain for the page and iframe source (respectively) are the same.

Creating an Auto Login

Let's look at how you can create an auto login such that users can click on a link in their portal and jump directly into this Item Maintenance program.

Begin the process of creating an auto login by navigating to the program that you want to allow your users to automatically log in to.

Then use the Tools>Auto Logins menu option to open this window:

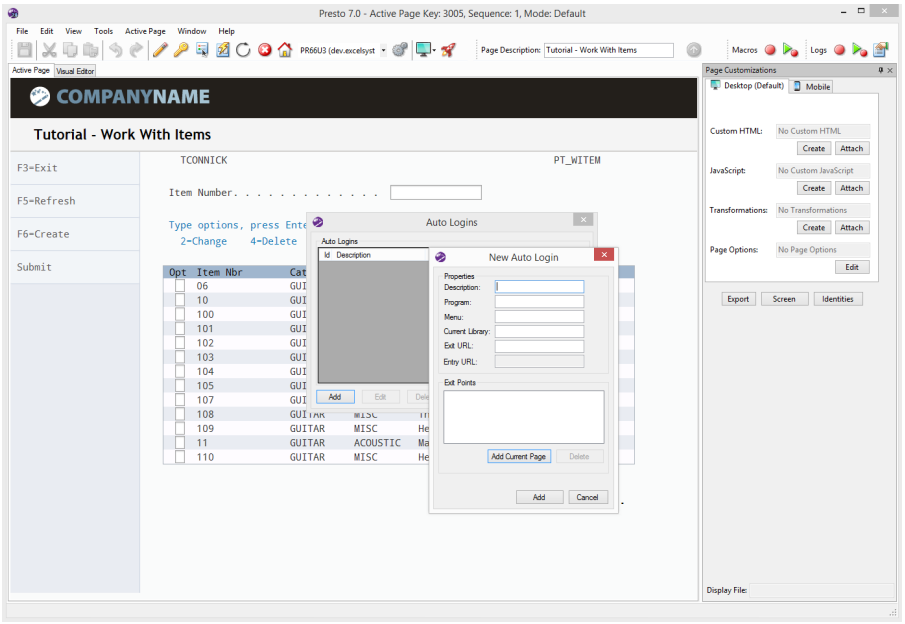


Figure 2-18. Creating an auto login in the Presto Designer.

In the window that appears, click the Add button to create a new entry.

These are the required values:

Description
Enter a description for the autologin entry.

Program
Enter the program name.

Menu
If you want the AutoLogin to go to an initial menu, enter the name here.

Current Library
Enter the program or menu's library.

Exit URL
This the URL of the program or page that the browser will be redirected to when the user backs up to the entry point and then attempts to exit from there (see Exit Points, below).

The Exit URL can be a generic screen in your corporate portal or website. In this way you can set up your portal to allow users to jump right into an IBM i application, and on exiting out, return to the originating web page on the portal, rather than an IBM i menu, command line or calling program.

The Exit URL must begin with “http://”.

Entry URL

The Entry URL is the link used to access the autologin. You can paste this value into Nexus if you are adding a link to this program in the Nexus ECM.

Exit Points

Exit Points are a way to execute an Exit URL (see Exit URL below) for the Autologin. You can have as many Exit Points as you need for each AutoLogin. For instance, if in your IBM i application there were various points from which a user could back out to a command line, you can add an Exit Point for that command line screen, and any time the user took an action to take them there, Presto would instead redirect them to the Exit URL.

How to Use Auto Logins

The basic premise of auto logins is to provide your users with a URL from which they can run their programs, for example:

```
http://192.168.0.100:8022/presto/presto?lnknbr=1&StrUpAPI=nexusaccess
```

The challenge for an auto login is that you must have a User ID and password.

Using Nexus

As you can see from our example URL above, one solution is to use Nexus 3.36 or higher as your web portal. Then you can handle the login process via the Nexus security framework of groups and users. If you are using Nexus, a Presto autologin program can retrieve the User ID and password that were used to log into Nexus.

In order to use this technique, there are number of requirements:

1. You need to have Nexus set up to have users sign in using IBM i user profiles.
2. In the Nexus Manage Site>Site Settings options, the 'Save IBM i Passwords' option needs to be checked.
3. You need to copy the Entry URL value for your Presto auto login and use that value as your link in Nexus. To copy this value, highlight the field (in Presto) and then right-click the highlighted value. Select copy, and then paste it into a new link in the Nexus ECM.
4. Finally, add '&StrUpAPI=nexusaccess' to the end of that URL.

To make all this work, Presto ships with an autologin program named LOGINAPI. This program does all the necessary things to work with Nexus and auto logins.



You can find instructions on how to make your Presto autologins work after upgrading to Nexus 4+ in our online Knowledge Base article 9710:

https://www.myfreschesolutions.com/kb?kbnum=9710

Please call Technical Support if you have any questions.

Using Custom Login Applications

If you don't have Nexus you can also create your own auto logins if you have a way to handle the security side.

When you install Presto, we include with the product the source member XL_PRESTO/QRPGLESRC LOGINAPI (your own current library may have a name other than XL_PRESTO). This is a template you can use to build your own login program. If you modify this source you can then rename the existing LOGINAPI program and replace it with your own.

If you take this approach, there are a number of important considerations:

1. The modified login program must be named LOGINAPI and exist in the environment you are running out of (i.e. XL_PRESTO) .
2. You cannot change the attributes of any of the parameters passed in or out of this program. You must stick to the framework set out in the template.
3. Your program must return a valid User ID (UserID) and password (PassWrd), in plain text.

There is a parm called '&StrUpAPI=' on the autologin URL. Use this to specify a cookie name that is retrieved, and from which 256 char of data are passed into your modified LOGINAPI program. While this is not a requirement, it means that you can pass details from a current browser session to LOGINAPI.

For example, in Nexus, this section of your URL string looks like this: '&StrUpAPI=nexusaccess'. The nexusaccess cookie contains the user number for the current user that has logged into Nexus. This is passed to the LOGINAPI we ship with the software, and we then use that number to chain to the User Master file and get the password. You can use a similar approach yourself.

The section "More About the Auto Login Template Shipped with Presto" on page 92 below includes additional technical details.

Library List Considerations

In order for this feature to work, you need to ensure that you have the correct library list set up. If you intend to use a custom login you need to create a data area named PRLIBL in your production environment, and then add your Nexus product library name to this data area.

Here is a version of the command that you can use:

```
CRTDTAARA DTAARA(XL_PR100/PRLIBL) TYPE(*CHAR) LEN(256) VALUE('XL_PR100 XL_NEXUS XL_
WEBSPT')
```

where XL_PR100 is the current product library and XL_NEXUS is the current Nexus product library.

Here is an example of what the data area should look like:

```
Offset *...+....1....+....2....+....3....+....4....+....5
0 ' XL_PR100 XL_NXTRUNK XL_WEBSPT '
50 ' '
100 ' '
150 ' '
```

Each library is 10 characters and they are listed in the data area at 11 character intervals. Use the CHGDTAARA command to edit the list.

Security Considerations

While in the program LOGINAPI you could in theory just hard-code a user name and password to make your auto login work, this would obviously represent a security issue. Nexus ensures that the cookie with the user login information is only created in a secure environment, and uses cross reference information to then retrieve the user profile information from another file. However, if you code your own approach for the LOGINAPI, you need to have an understanding of the security issues involved.

More About the Auto Login Template Shipped with Presto

Presto ships with an auto login template that you can use to retrieve and then pass Presto your user profile and password. The template is an API, the source for which is in XL_PRESTO/QRPGLESRC LOGINAPI.

The default program shipped with Presto is configured to receive a link number and login cookie name (nexusaccess). From that cookie it retrieves the user profile stored by Nexus, and returns that to Presto so the auto login can be executed. If you have your own portal or mechanism for your users to sign in, you'll need to create and populate a cookie with whatever information LOGINAPI will use to retrieve the actual IBM i user profile and password.

These are the parameters you need in LOGINAPI:

Parameter	Description
Login Cookie Data (strupapi)	This optional parameter allows you to specify a cookie name, containing your initial URL. This can contain up to 256 characters.
Link Number (lnknbr)	This is part of the URL, and retrieved from PR_AUTOPF.
User Profile & Password	The user profile and password are returned from the login API.

Here is an example of URL string you could either create a link to, or provide to your users:

```
/presto/presto?lnknbr=1&StrUpAPI=nexusaccess
```

(this would be prefixed by your domain name or HTTP server IP address and port number). In this example, we are invoking link number 1 (the first auto login we created), and as a second parameter, passing the content of the cookie 'nexusaccess'.

Then LOGINAPI retrieves the link and cookie information, executes the code to get the user profile and password, and returns those values to Presto, which then uses them to complete the login process.

Spool File Listings

Presto features a way for you to easily download your IBM i spool files to your PC. Press Ctrl+Alt+P to download a list of spool files similar to this:

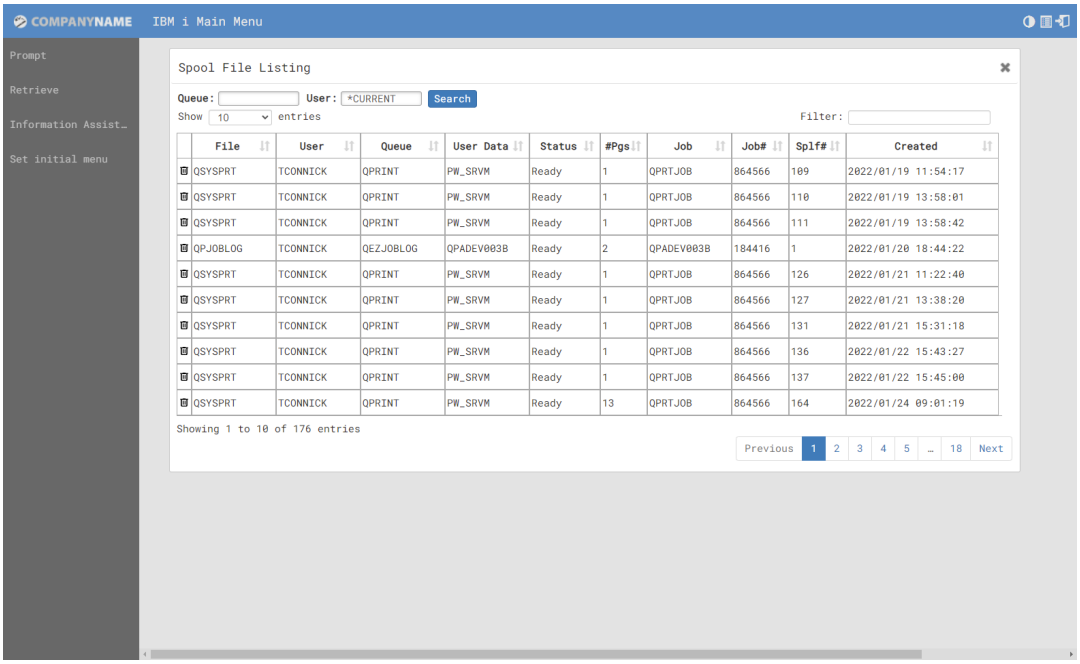


Figure 2-19. Reviewing a list of spool files after pressing Ctrl+Alt+P.

This feature is available to all users who have *SPLCTL special authority.

You can sort this list by clicking on the column headings or use the controls at the bottom to navigate forward and backward through the list.

To open a spool file as a PDF on your PC, double click the entry in the list above. Below is an illustration of a downloaded file:



Figure 2-20. Reviewing a downloaded spool file PDF in a browser.

From here you can save the file to your PC, or print it.

Support for AFPDS spool files. You can now view AFPDS spool files in PDF format (note that view as Html is not supported). This requires a minimum OS of 7.1, with PTF SI43471. For details, see https://www.ibm.com/support/pages/using-cpysplf-re-create-spooled-file-or-create-pc-file in IBM Support.

IASP Support. The Spool File Viewer will now display Spool Files from all IASPs in your system.

Technical Details

The spool file download and display feature is implemented via these events:

- SplfListEvent.Open
- SplfListEvent.Close

These fire immediately before the named action, and returning false from them prevents the action.

Calling the Functions Manually

You can implement this feature manually using the function DisplaySplfDialog(oFilterData). For example:

```
<a href="javascript: DisplaySplfDialog({"User":"*CURRENT"});">Show Spool Files</a>
```

You can load the spool file list manually using any combination of the User, System, FormType and UserData. Some restrictions do apply:

- if you want to specify an OutQueue, you must specify both OutQLib & OutQ
- if you want to specify a job, you must ONLY specify Job, JobNum & User

In addition to (or instead of) the above, you may also specify creation time filters. The four creation filters are StartDate, StartTime, EndDate & EndTime.

You may specify:

- StartDate = "*ALL"
- StartDate = "*FIRST"
- StartDate = CYYMMDD
- StartTime = HHMMSS
- EndDate = "*LAST"
- EndDate = CYYMMDD
- EndTime = HHMMSS

Here is an example:

```
DisplaySplfDialog({"User":"FRED", "UserData":"PRESTO", "StartDate":"1130101"});
```

This will return a list containing all of Fred's spool files with user data PRESTO that were created on or after Jan 1, 2013.

Adding a Site Favicon

Presto supports the ability to add a favicon to your site. A favicon is "is a file containing one or more small icons, most commonly 16×16 pixels, associated with a particular Web site or Web page"[1].

In other words, the little picture that you sometimes see to the left of the URL in your browser's address bar.

To add a favicon to your Presto site, follow these instructions:

1. Add this line to your skin's layout.html file:

```
<link rel="SHORTCUT ICON" href="images/favicon.ico">
```

2. Add your favicon.ico to your skin's images folder.

You can perform both of these operations from the Presto Designer's Tools>Modify Skin menu options.

[1] http://en.wikipedia.org/wiki/Favicon

Creating a New File

Presto supports the ability to create a new file for your site skin. You can perform this option from the Presto Designer's Tools>Modify Skin menu option.

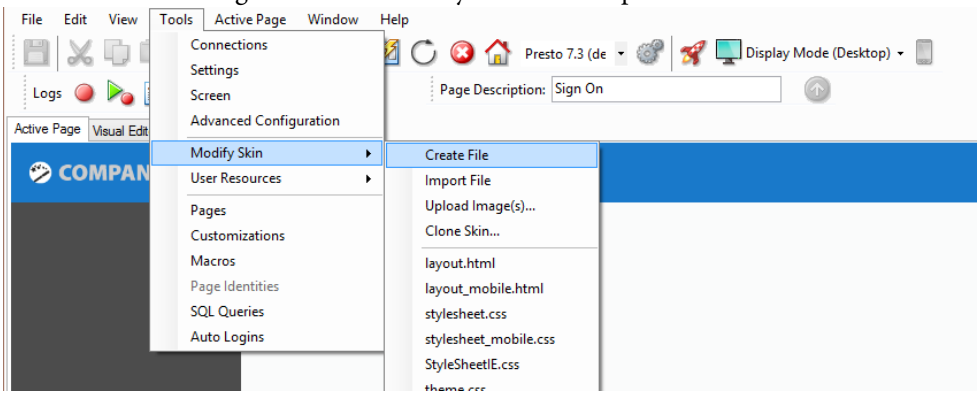


Figure 2-21. Create a new file.

Presto displays this prompt to enter the file name:

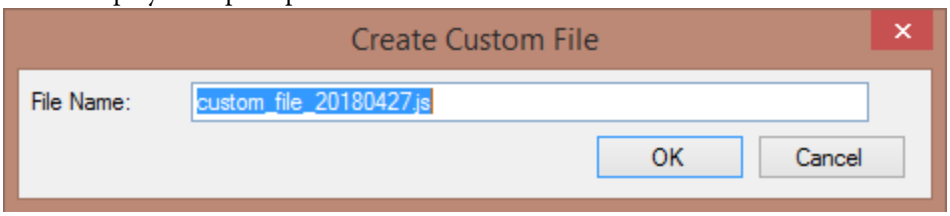


Figure 2-22. Entering the file's name.

After entering the file name, the file opens in the Presto designer, with this generated comment:

```
<script language="JavaScript" type="text/javascript" src="skins/Pacific/custom_file_
20180427.js"></script>
```

Use this link to reference the new file in your Skin's layout.html.

Rotate Orientation of Display Mode

In Presto 7.1, we added support for previewing mobile devices with a variety of screen sizes. Presto now provides the option to preview devices in both portrait and landscape orientation.
Portrait Orientation:

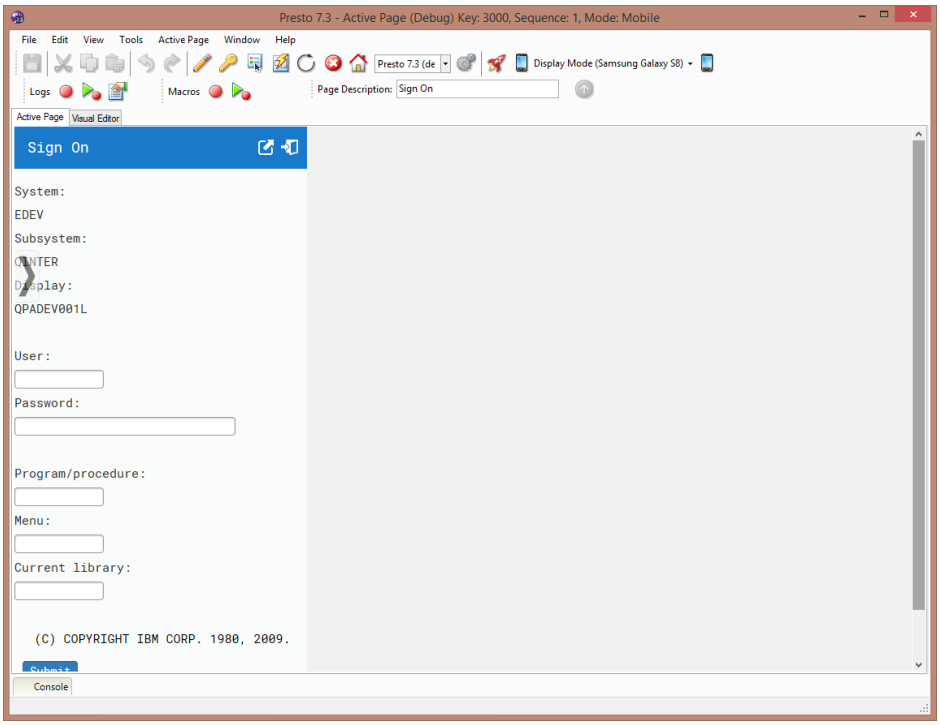


Figure 2-23. Previewing the display mode in portrait orientation.

Landcape Orientation:

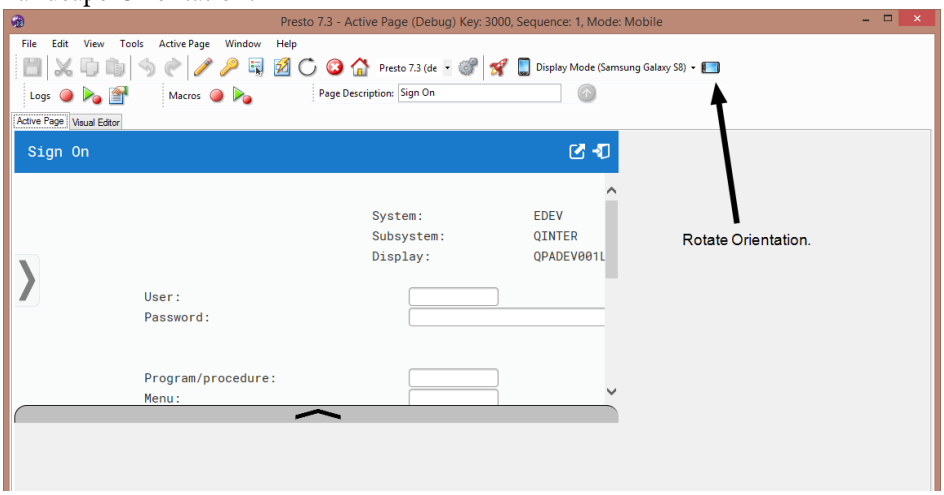


Figure 2-24. Previewing the display mode in landscape orientation.

Access PC Applications from Your Presto Site

While there are high walls between browser pages and PC apps on the underlying system, in some cases it is very beneficial to allow IBM i applications to interact with PC applications. For example, we have customers with imaging applications where users can select records in RPG programs and load related pictures or PDFs on their PC.

Requirements

Below are the requirements to enable this feature.

STRPCCMD Access

In order for the user to use this feature, they need IBM i authority to run the STRPCO (Start PC Organizer) STRPCCMD (Start PC Command) commands.

Install the StrPcCmd Utility on the User's PC.

Your Presto IDE installation includes the StrPcCmd Utility in a folder by default installed to C:\Program Files (x86)\ESDI\Presto 10.0\StrPcCmd Installer.

Double-click the install executable file PrestoStrPCCmd_Installer to install the utility on your PC.

Enable this feature in Presto.

Edit the Presto Advanced Configuration options (Tools menu) and enable the Start PC Command option:

```
<EnableStrPcCmd>1</EnableStrPcCmd>
```

Set the port on which the StrPcCmd Utility is running:

```
<StrPcCmdPort>5000</StrPcCmdPort>
```

This port should match the Listen Port in the StrPcCmd Utility:
downloaded file:

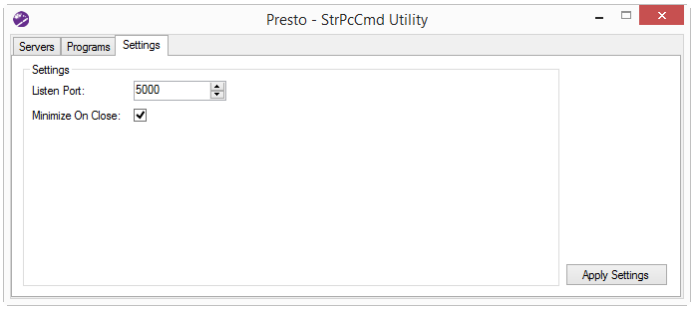


Figure 2-25. The StrPcCmd Utility.

How to Use this Feature

In order to use this feature, run these steps:

1. On the server the run STRPCO command.

2. Prompt the STRPCCMD with F4. Enter the command you wish the execute. If the path contains spaces, you must wrap it with parentheses. For example: STRPCCMD PCCMD('"C:\Program Files (x86)\ESDI\Presto 10.0\Presto.exe"').

HTTPS and StrPCcmd

StrPcCmd does not support HTTPS encryption but it can still work with HTTPS Presto setups and, depending on the target browser, will require a few simple steps to make it work.

```
<meta name="referrer" content="unsafe-url">
```

Chrome and Firefox

Modify **layout.html** of the skin you use and add the following code to **HEAD** tag

```
<meta name="referrer" content="unsafe-url">
```

Internet Explorer

It will require **"mixed content"** to be allowed in the Security Settings

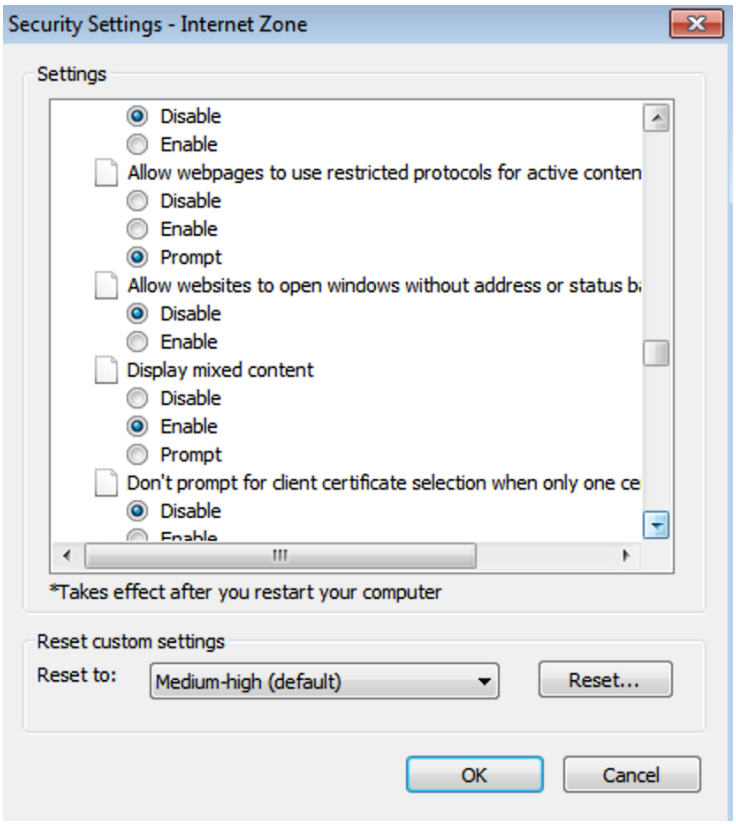


Figure 2-26. Internet Explorer Security Settings



StrPcCmd allows the server to open applications on a user's PC. While the user must approve both the server issuing the request and the command to run, this is still considered a vulnerability.

Further, using the meta tag described for HTTPS and for Chrome and Firefox causes all HTTP requests and links to the outside world to leak information about the Presto environment's URL.

We recommend clients consult their IT security experts before taking steps to enable StrPcCmd.

Regular Expressions

Here are some troubleshooting suggestions:

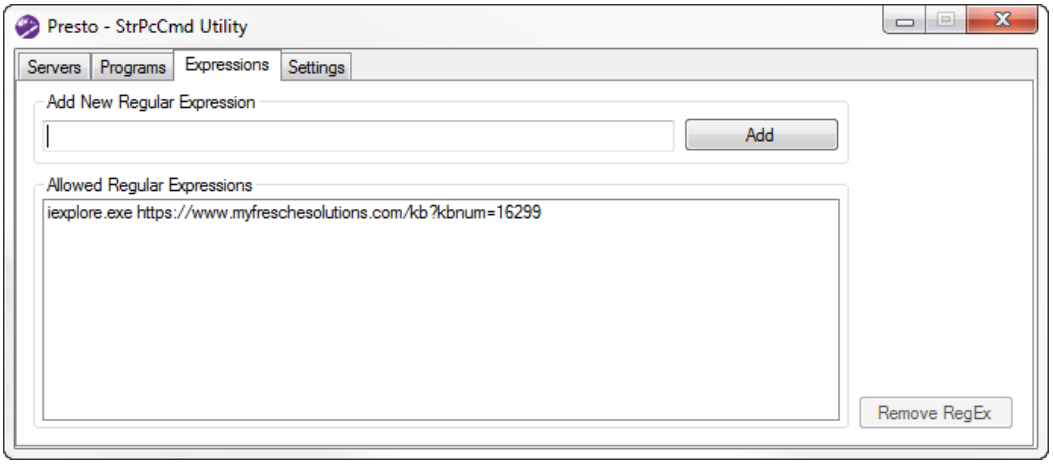


Figure 2-27. Regular Expressions

The Expressions tab of the StrPcCmd Utility provides the ability to allow all versions of a command regardless of the passed parameters.

For an example of where this is used, consider that there is a program that users run from the green screen that opens a link to a selected Knowledge Base article in Internet Explorer. The program call could look like this:

```
iexplore.exe https://www.myfreschesolutions.com/kb?kbnum=16299
```

The Utility will store that exact string as the allowed command. If you open a different article, the Utility will again pop up asking you whether you want to allow the command to run. This can get tedious if users are constantly opening different articles. By providing a Regular Expression like this:

```
iexplore\.exe https:\/\/www\.myfreschesolutions\.com\/kb?kbnum=\d{5}
```

This can allow all links to Knowledge Base articles while still getting the security warning for any other use of Internet Explorer.

Import/Export

The Settings tab of the StrPcCmd Utility provides buttons to import and export the user configuration.

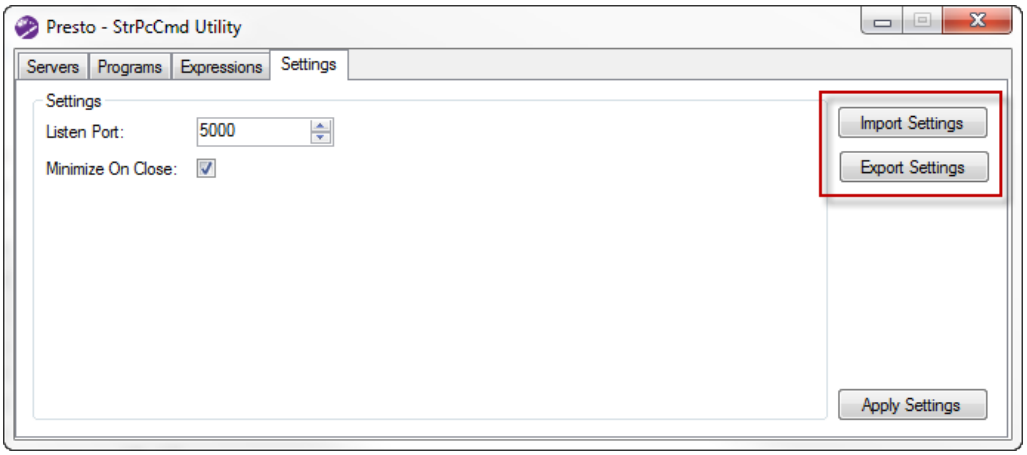
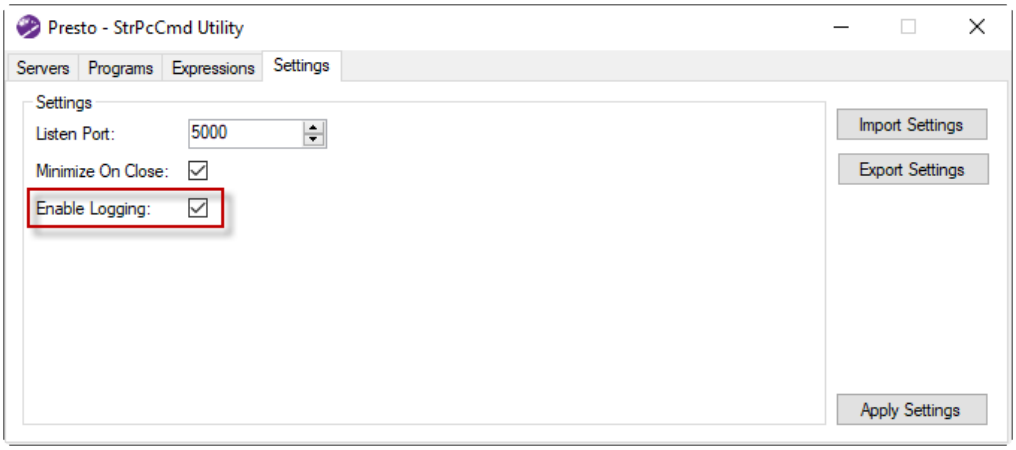


Figure 2-28. Importing and Exporting Settings

With this option, users can configure the Utility once and then apply that configuration on other PCs.

Logging

The Settings tab of the StrPcCmd Utility provides the option to enable Logging:



Once enabled, the logs will be written to a file located at C:\Users\[User]\AppData\Roaming\ESDI\Presto\PrestoStrPcCmd-[current date].log.

These logs contain detailed information such as the start and stop of listing on a port, handling incoming HTTP requests, interacting with the server, and executing external programs and their results.

Troubleshooting

Here are some troubleshooting suggestions:

Symptom	Suggested Action
You receive an error "STRPCCMD use has been disabled. Please contact your admin."	In your configuration settings, confirm the option is enabled: <EnableStrPcCmd>1</EnableStrPcCmd>
Unable to access StrPcCmd. Please	Most likely the StrPcCmd utility is not running.

Symptom	Suggested Action
start StrPcCmd program and verify your configuration.	If it is running, then this may indicate a broken or incompatible version.
You received an error: "We are unable to validate Origin and Referer."	Most likely your server is using **HTTPS**, this case modify **layout.html** file in your skin to include the following code in HEAD section: <meta name="referrer" content="unsafe-url">

Server-Side Whitelist

Presto also provides a method for whitelisting specific commands on the server side.

This is handled via the file `strpccmd_whitelist.txt` found in `/esdi/xl_presto` on the IFS.

If this file is empty or missing, all commands will be allowed. This is the default.

If you add text to the file, then all commands will be compared against the contents of the file and only commands that match will be allowed to run.

For any command that does not match the whitelist, the text **"*Command not allowed due to Whitelist"** will be passed to the Utility instead of the command.

Commands are matched to the whitelist with parameters and path information stripped off.

If you want to allow Internet Explorer for example, you would include **iexplore.exe** in the whitelist.

On calling StrPcCmd, you could then specify the command as **iexplore.exe http://google.ca** or **"C:\Program Files (x86)\Internet Explorer\iexplore.exe" http://google.ca**.

Note: In the second example, we need to wrap the command in double quotes when we call it due to the spaces in the folder path. If the folder path contained no spaces, the double quotes would not be necessary.

The double quotes can also be used to group some parameters with the command name so as to include them as part of the command in the whitelist.

For example, if you wanted to allow opening Internet Explorer to only a specific web page, you could add **iexplore.exe http://google.ca** to the whitelist and then when calling it you would need to use the double quotes as in **"iexplore.exe http://google.ca"**.

Whitelist entries should be separated by new lines:

Example Whitelist

```
iexplore.exe
chrome.exe
notepad.exe
```

Overriding the default StrPcCmd Handler

Presto has the ability to override the default StrPcCmd functionality via a JavaScript Event Handler.

If you wish to use StrPcCmd but is unable to use it due to security rules, you can bind a Handler to the STRPCCMD event to intercept the command and process it. The handler will be limited

by the fact that JavaScript runs in the browser, and this environment is secured from the rest of the PC. However, the handler can still open URLs and trigger requests to external APIs in some cases.

Here is an example StrPcCmd Handler:

```
jQuery(document).on("STRPCCMD", function(e, cmd, pause) {
    console.log(cmd);
    console.log(pause);
    window.open(cmd.substr(13), "_blank");

    if (pause)
    {
        console.log("paused");
        let i=Date.now();
        while(Date.now() < i+10000)
        {}
        console.log("unpaused");
    }
    return false;
});
```

This example Handler assumes the command is iexplore.exe followed by a URL. It will open the URL in a new tab. If the pause variable is set, it will wait 10 seconds before returning.

Returning false (or setting e.PreventDefault) from the Handler will prevent the default functionality from running and will return control to Presto.

Returning true from the Handler will allow the default functionality to run as normal.

The parameters accepted by the function:

- **e** - The event object
- **cmd** - The string containing the command and any parameters. For example iexplore.exe http://google.ca. If the command entered was blocked by the Server-side Whitelist, this will contain *Command not allowed due to Whitelist.
- **pause** - A boolean variable. If the PAUSE parameter to StrPcCmd was *YES then this will be true. Likewise pause will be false if *NO.

Dual Maintenance Management

Presto now allows you to track changes in Display Files associated with customized screens.

The Presto designer shows customized screens that may have had the underlying DDS changed. As a result, you can go to these screens that have DDS changes to verify the customizations are still working.

To configure the detection for display files, open the Presto designer and go to Tools>Advanced Configuration. Find the following element:

```
<EnableDspfDetection>1</EnableDspfDetection>
```

This option is enabled (1) by default.

How does it work?

The display file is displayed at the bottom of page customizations:

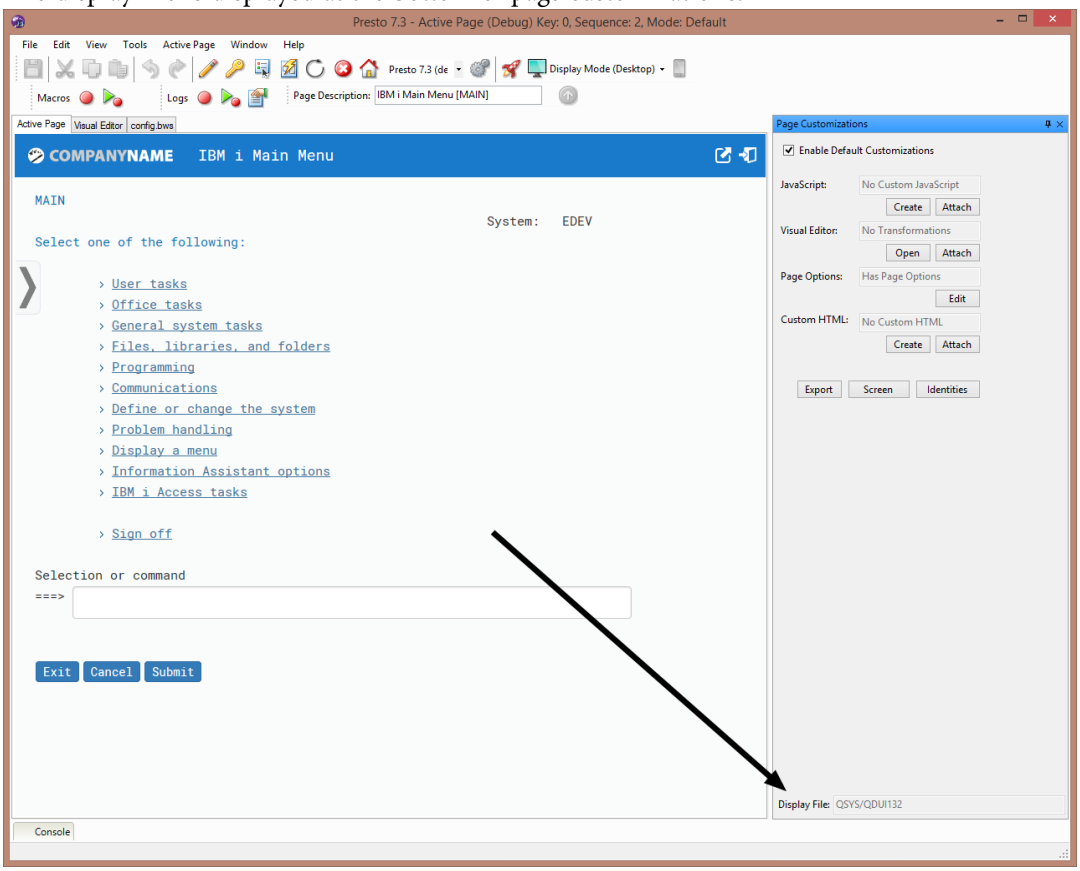


Figure 2-29. Display File associated with a screen.

Presto displays the path to the Display File:

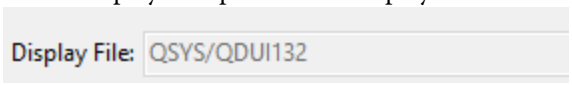


Figure 2-30. Display File for current screen.

Presto will detect when a display file is changed and display a notification.

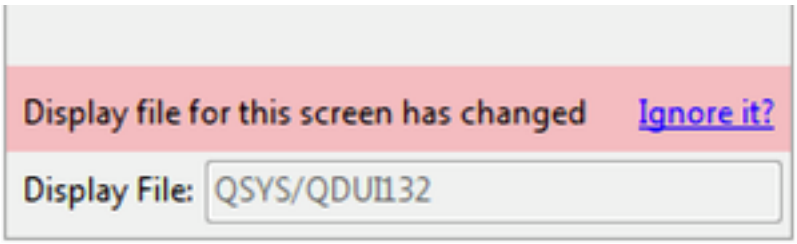


Figure 2-31. Notification after changing display file.

Users have the option to either modify customizations to reflect changes to the Display File or to click Ignore it.

Working with Display-File Changes

In the Presto designer, go to Tools>Pages to work with all the changed Display Files.

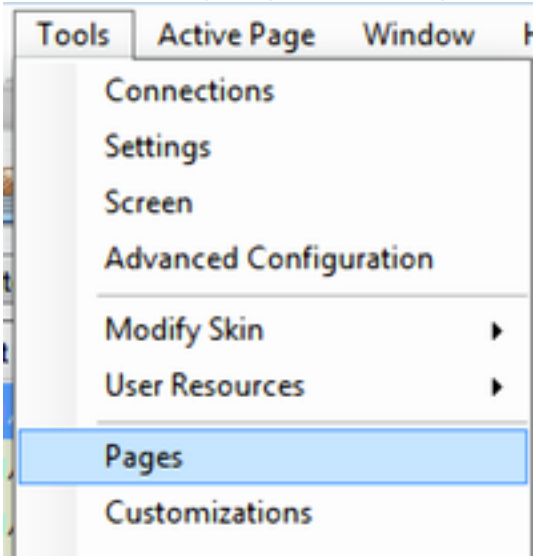
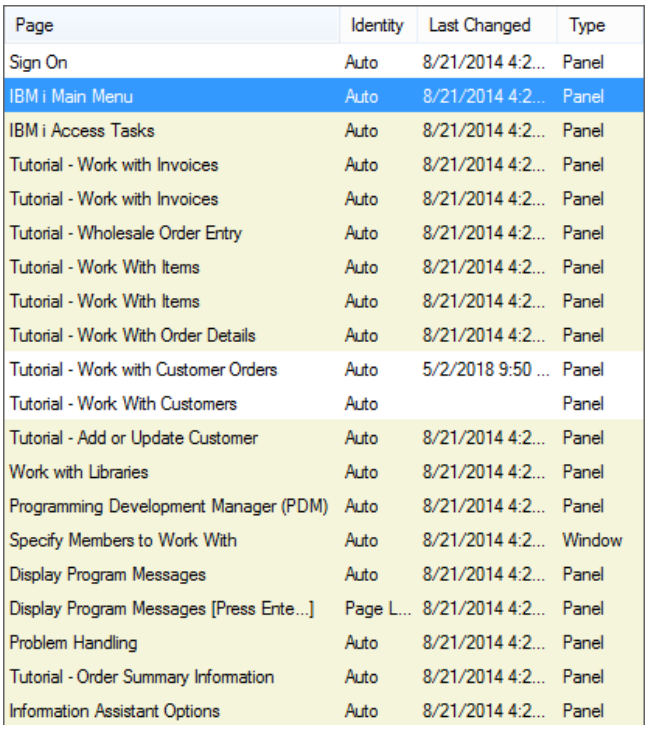


Figure 2-32. Working with display files.

Presto highlights the customized screens with modified display files.



Page	Identity	Last Changed	Type
Sign On	Auto	8/21/2014 4:2...	Panel
IBM i Main Menu	Auto	8/21/2014 4:2...	Panel
IBM i Access Tasks	Auto	8/21/2014 4:2...	Panel
Tutorial - Work with Invoices	Auto	8/21/2014 4:2...	Panel
Tutorial - Work with Invoices	Auto	8/21/2014 4:2...	Panel
Tutorial - Wholesale Order Entry	Auto	8/21/2014 4:2...	Panel
Tutorial - Work With Items	Auto	8/21/2014 4:2...	Panel
Tutorial - Work With Items	Auto	8/21/2014 4:2...	Panel
Tutorial - Work With Order Details	Auto	8/21/2014 4:2...	Panel
Tutorial - Work with Customer Orders	Auto	5/2/2018 9:50 ...	Panel
Tutorial - Work With Customers	Auto		Panel
Tutorial - Add or Update Customer	Auto	8/21/2014 4:2...	Panel
Work with Libraries	Auto	8/21/2014 4:2...	Panel
Programming Development Manager (PDM)	Auto	8/21/2014 4:2...	Panel
Specify Members to Work With	Auto	8/21/2014 4:2...	Window
Display Program Messages	Auto	8/21/2014 4:2...	Panel
Display Program Messages [Press Ente...]	Page L...	8/21/2014 4:2...	Panel
Problem Handling	Auto	8/21/2014 4:2...	Panel
Tutorial - Order Summary Information	Auto	8/21/2014 4:2...	Panel
Information Assistant Options	Auto	8/21/2014 4:2...	Panel

Figure 2-33. Displaying customized screens with modified display files.

If you're sure there's no required changes to your Presto customizations, you can select the screens and click the Ignore button at the bottom of the window.

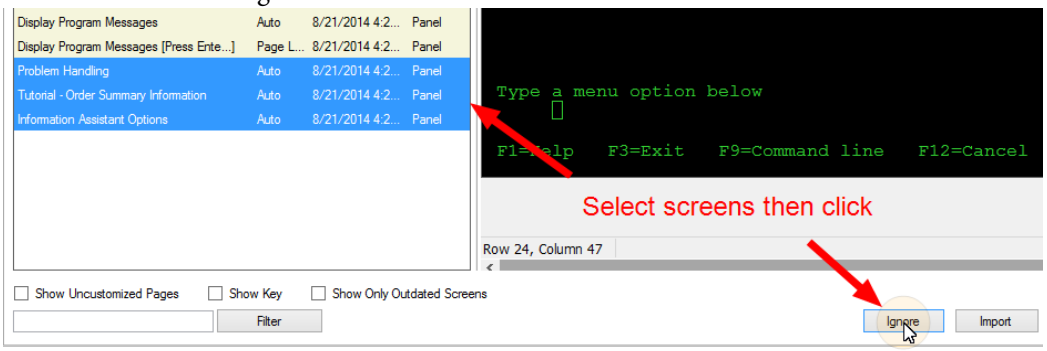


Figure 2-34. Ignoring changes in the display files of selected screens.

The Ignore button marks the selected screens to ignore DDS changes.

Show Only Outdated Screens

To review all the screens with DDS changes, check the 'Show Only Outdated Screens' option on the bottom left of the manage pages window.

Page Up and Down Buttons

Presto now displays Page Up and Page Down buttons at the bottom-right of a detected subfile.

Presto autodetects these elements on your screen using a detection pattern. You can configure the detection pattern (Detection is based on the More and Bottom elements text) in the configuration file config.bws (Tools>Advanced Configuration). Here is the default settings currently defined:

```
<module name="SubfilePaging" enabled="1" />
   <parameter name="MorePattern">More\.{3}</parameter>
   <parameter name="BottomPattern">Bottom</parameter>
</module>
```

The option is enabled by default. To enable, set the `enabled` attribute to '1' (set to '0' to disable).

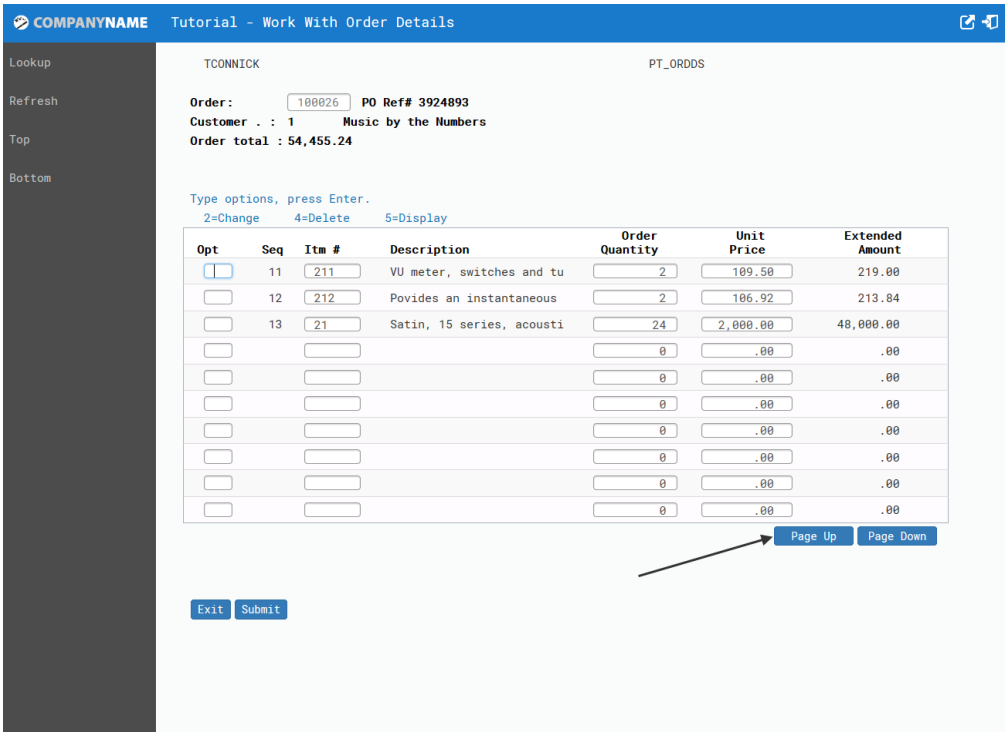


Figure 2-35. Page Up and Down Buttons.

Submit, Cancel, Exit Buttons

Presto now displays the Submit, Cancel, and Exit options as buttons at the bottom left.

These buttons are autodetected on your screen using a detection pattern. You can configure the detection patter in the configuration file config.bws

```
<module name="NavigationButtons" enabled="1">
   <parameter name="Selector">[funckey=03],[funckey=12],.submit</parameter>
</module>
```

The option is enabled by default. To enable, set the `enabled` attribute to '1' (set to '0' to disable).

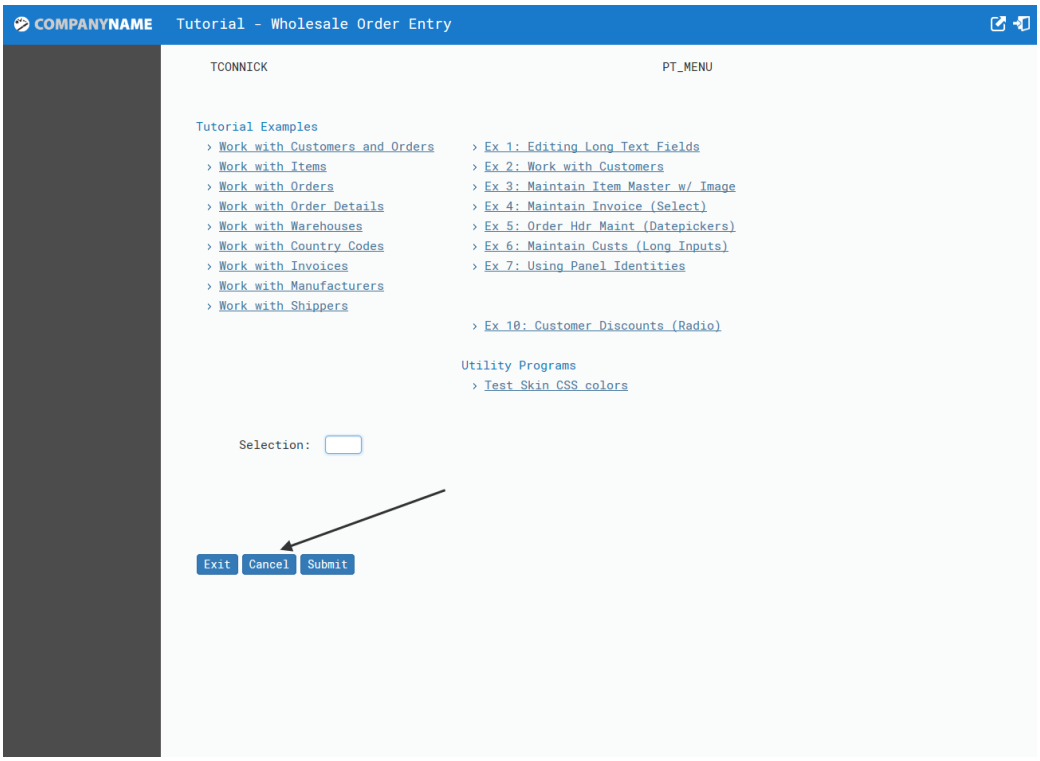


Figure 2-36. Exit, Cancel, and Submit.

Window and Table Text

The Presto designer shows new text on the right panel to display the window and table dimensions.

The window and table dimensions of the active page are displayed at the bottom of page customizations:

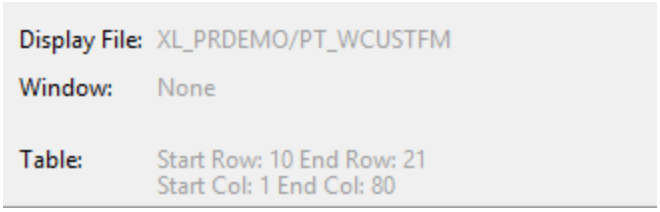


Figure 2-37. Window and Table Dimensions associated with a screen.

Unique Naming of Customizations

Presto will now use unique customization names for each screen and won't allow customizations of the same type (e.g. HTML, JavaScript, or Transformations) to have exactly the same name.

It will also store the contents of the customization with the file name matching the customization name, meaning that if you create a customization and give it the name `Sign On` then the file will be called `Sign On.js`.

This supports change management processes with external version control systems. It allows developers to work in separate Presto environments and store their page customization's files with unique names in a version control system.

Renaming a Customization

If you want to change the name of a customization, there are a few different ways that is possible from Presto.

- In the Page Customizations area of the Presto Designer, click in one of the inputs available and type your text to rename the customization:

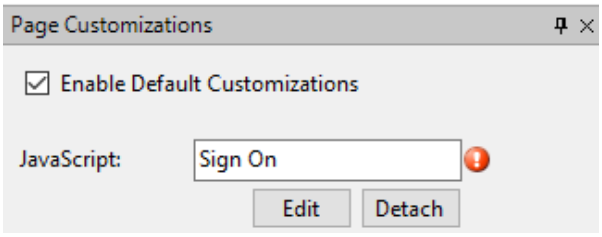


Figure 2-38. Page customization panel

- When you open a customization of any kind in the Presto Designer, you can right-click the file tab and select the Rename... option:

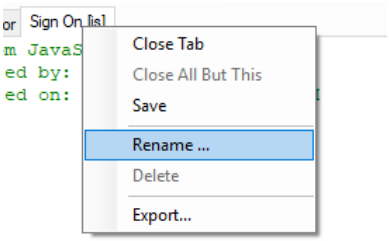


Figure 2-39. Preparing to rename a customization.

- From the Manage Pages window (Tools>Pages), you can work with all your pages and rename the individual customizations on those pages:

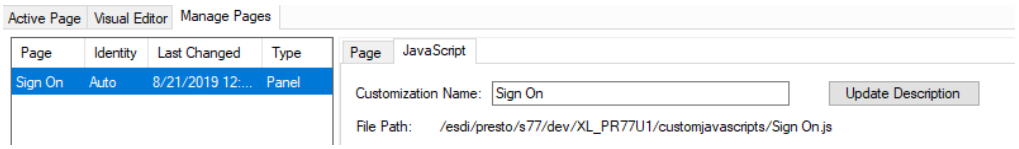


Figure 2-40. Renaming in Presto's Manage Page Interface.

You can also work with customizations (Tools>Customizations), and from there rename individual customizations. Here is an illustration of a customization accessed from that tab:

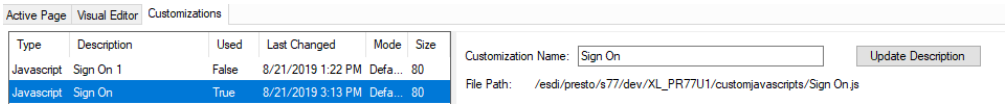


Figure 2-41. Renaming the customizations created in your environment.

The list on the left includes entries for every custom HTML, JavaScript file, and set of transformations you create. On the right, you can rename your customization name. Change the text and click the 'Update Description' button.

Naming Conflicts

If your new customization name matches an already existing customization then you will be asked to provide a different name:



Figure 2-42. New customization matches an already existing customization.

If the new name of the renamed customization matches an already existing customization then an exclamation sign will be displayed next to the field with a tooltip describing the error:

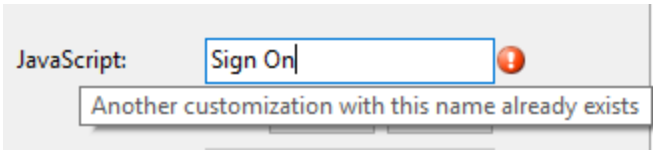


Figure 2-43. New customization matches an already existing customization.

To reduce the possibility of conflicting with any existing customizations, all mobile customizations have `[Mobile]` prefix added to their name and the default/desktop customizations will use the Panel Description by default.

Supported Characters

All characters are supported, but currently the customization will only be used as a file name when it matches the following regex: `^[0-9A-Za-z_\[\]\(\)\-\.+=;,`!?@#&%$~^]+$`

There is no identification of an error when other characters are used as Presto fallsback to using the numeric form `000000000NNN` for the file.

Customization Not Found

If a customization file is not found then a Not Found label in the Customization and Manage Pages UIs will appear:

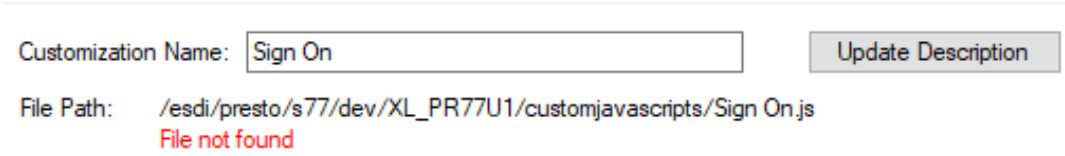


Figure 2-44. Customization File not found.

Export Customizations without the Contents

To support third-party methods to synchronize customization files between environments, it's possible to enable the display of the Include Contents of Customization files checkbox on Export via the `CRTL+F11` hidden settings form.

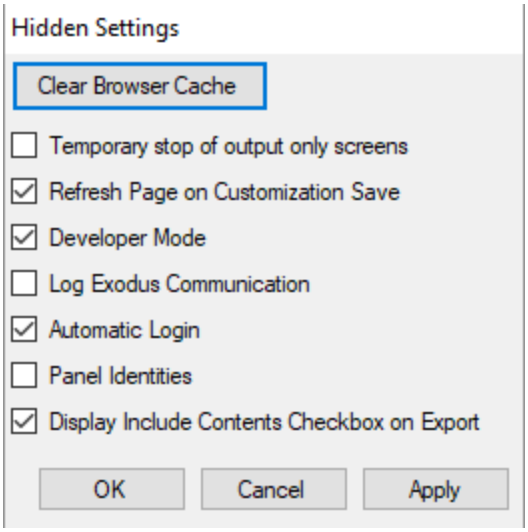


Figure 2-45. Include contents of customization files checkbox on Export.

Then during the export process you will have an option to uncheck **Include contents of customization files**:

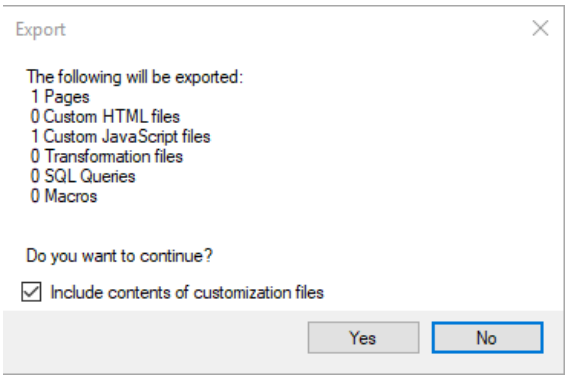


Figure 2-46. Export option for customization files.

Considerations for Upgrading and Existing Customizations

Customizations created in previous versions of Presto use numeric file names like `000000001.js`. They won't be modified unless you decide to rename them.

The same applies for possible name collisions in older environments, so they can continue to exist. But the new customizations won't allow the use of pre-existing names.

Auto Detection Settings and Identities

Now that you have learned about Presto's customization tools and techniques, you may feel ready to start adding features to your own pages.

But before you forge ahead there is a critical aspect to how Presto works with your applications that you should assess *after* customizing a few of your own screens but *before* you get too far along in that process: you need to evaluate how well the default auto detection settings are working and whether you need to adjust them to better fit your application's screen layouts.

In other words, when you add custom changes to a screen:

1. Will Presto recognize every variation of that screen to consistently apply your custom features at run time, every time?
2. Will Presto NOT confuse that screen with others, incorrectly applying custom features to the wrong page(s)?

As you start working with Presto the pages you should particularly test are:

- **Subfile programs (especially at the top & bottom of your lists).** The difference in a subfile, and particular at the start and end of your lists, can make those variations appear to be different programs.
- **Pages that conditionally display a variable number of fields.** Conditions may be based on the selected data or the user running the program.
- **DDS windows with atypical window border patterns.** Some window border patterns may be difficult to distinguish from the underlying screens.

While each of us may believe our own design standards are pretty 'normal' there is actually a dizzying range of 'normal' in the IBM i world! This chapter will help you understand this aspect of your Presto configuration and teach you how to adjust the software to work best with your standards.

Understanding Your Detection Settings

Presto ships with these default Detection Settings (Tools>Settings>Screen Detection):

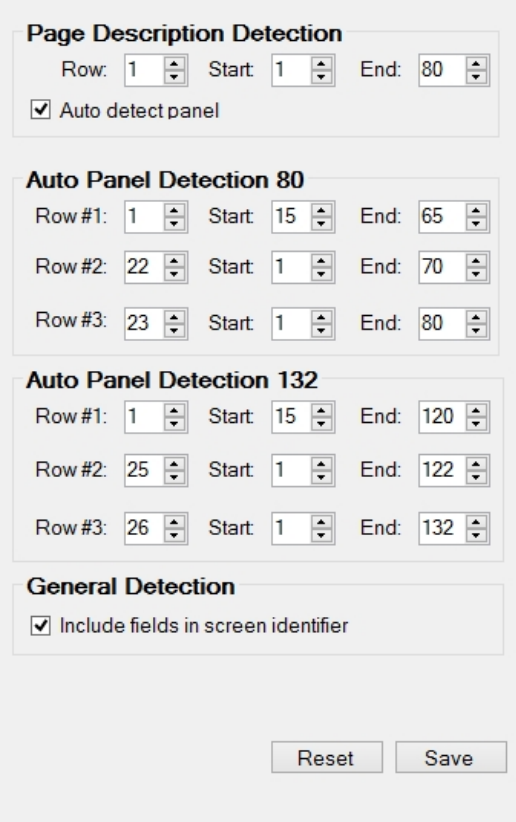


Figure 2-1. The automatic panel detection settings.

The auto panel detection settings drive the way in which Presto distinguishes between screen in order to create, save and consistently apply changes and customizations.



If you are using Presto to render your site entirely using the 'out of the box' settings, or global customizations, then it's not important for the software to be able to distinguish between screens and cross reference to the customization files - Presto will just present screens as it finds them.

In practice, however, almost everyone is customizing at least some screens.

The options in Figure 2-1 are discussed in detail in the section "Detection Settings" on page 40, but here is a brief summary:

- **Page Description Detection.** This allows you to set the location from which Presto will pull your initial page descriptions. As you add customizations to your screens it's important to make sure they also have meaningful and reasonably unique descriptions, or it can be challenging to manage your changes.
- **Auto Panel Detection Settings (for 80 and 132 column screens).** These are the locations that Presto will scan for unique content (typically a page title and function keys) to use to identify the screen. We generally recommend using rows 1, 22 and 23, but with some applications other rows may work better.
- **Include Fields in Screen Identifier.** It's usually good to also include the fields on the screen in the set of information used to uniquely identify the page. The exception to

this recommendation is if a high proportion of your screens include conditionally presented fields.

These options are discussed in more detail in the following sections.

Detection Settings and OA

Because of how OA programs work, Presto does not use these detection settings for OA programs. Instead, Presto uses a combination of the record format names and field names.

Why Use 'Auto' Detection Settings?

In creating Presto we deliberated at length about whether to design the software to identify screens automatically vs. requiring developers to individually mark and tag every screen they customized. In the end we chose an approach that attempts to automatically identify the majority of your screens, while still allowing you to override this process with global identities (more on this later).

The software runs most efficiently when you rely primarily on Presto's ability to build and assign unique IDs for each screen. The alternative, of manually 'tagging' each and every panel, takes extra time during your development phase and also carries a minor performance penalty at run time.



This performance penalty is small, but if your set of customized screens grows into the hundreds or thousands, the performance impact may start to be non-negligible.

How Auto Panel Detection Settings Work

To visualize the impact of the settings in Figure 2-1, consider this illustration:

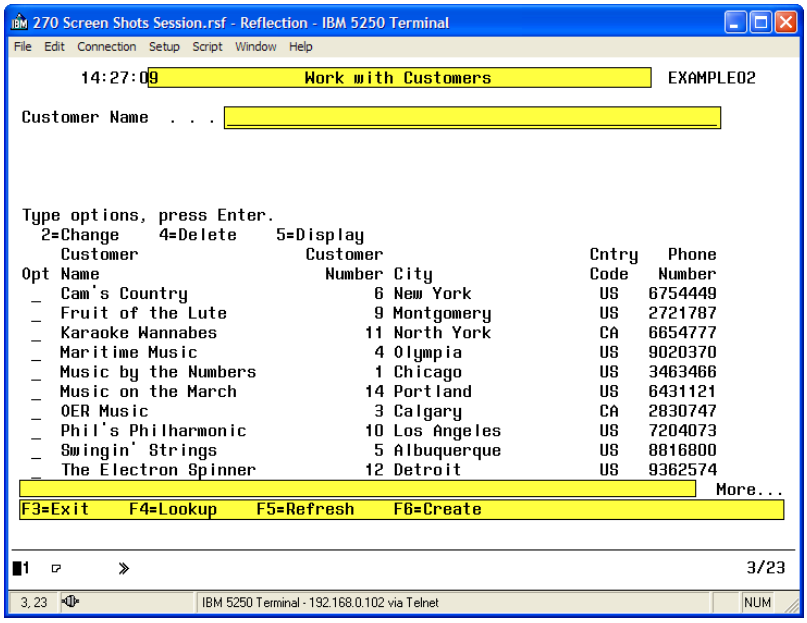


Figure 2-2. A typical IBM i subfile panel.

The yellow areas correspond to the default 24x80 auto detection settings. As you navigate through your applications, for all standard panels (windows are handled differently) Presto looks at the areas highlighted above:

- A swath across the middle of the first row (Row 1, columns 15 to 65).
- Input fields that are NOT in the subfile (in this example, the Customer Name field).
- Row 22, columns 1 to 70, which will often have function keys.
- Row 23, columns 1 to 80, which also usually have function key text.

Notice that the default settings avoid an anticipated More.../Bottom message at the end of row 22.



You can display your own panel detection settings using the button at the bottom of your green screen view in the Presto Designer. Click the 'Screen' button on the right side of the designer and press the 'Show panel detection areas'. Note that this preview doesn't include the fields, just the static areas.

Window Detection

Windows are identified differently than standard panels.

When Presto finds a window on a screen it identifies it using text from the top and bottom borders. This generally means the window title and function keys.

Potential Trouble Spots

Here are some issues to watch out for.

Program Changes

If you change a program, for example by adding or removing fields or function keys, Presto may no longer recognize that screen as the one to which you originally added custom features.

Solution

This is not unexpected and if it happens you can use the Attach buttons in the Presto Designer to reattach the HTML, JavaScript or Transformations. The section "Reattaching Custom Changes" on page 122 below illustrates this.

Variable Information in Auto Detection Rows

Most programs include a program title in the first row, but some may also include other fields that do vary. So Row#1 detection settings attempt to limit themselves to a section at the middle of the top line of your programs:

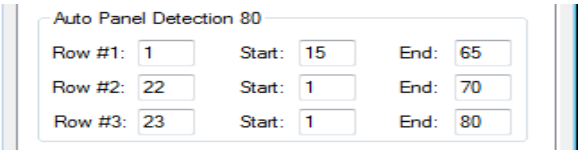


Figure 2-3. Auto detection settings to trap the program title.

The sample program illustrated in Figure 2-2 has the program title in the middle of row 1, but it also includes a Current Time field beside it. Notice in the illustration that the last digit of the

time field (the seconds) is in column 15, within the area Presto is using to determine whether this is a unique screen.

As users come to this program the first character in the auto detection area will include any value between 0 and 9. Consequently, Presto may internally identify and tag this panel up to 10 different times, depending on the precise second at which the program is run.

A Time field is just one example of variables that are often found in row 1. Other common offenders include dates, user profiles, site locations, department IDs, etc.

Solution

The solution to this common scenario is to adjust your panel detection settings for a particular row to exclude variable content.



The catch is that if you have variables scattered all over your headings, you may then find yourself trimming back your automatic detection settings to almost nothing, which doesn't work well either. More on this later!

Conditionally Presented Fields in Row 1 (or Elsewhere)

Presto normally uses non-subfile fields on your screen to help ID a page:

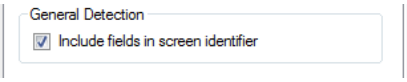


Figure 2-4. The option to include the screen fields in the identifier.

This option excludes fields in an area recognized as a subfile. Figure 2-2 includes one input field outside the subfile.

Some programs may have fields that appear only conditionally, based on which user is running the program or the particular circumstances under which the program is run. Because Presto can only see what is currently delivered by the datastream, rather than the actual DDS of the program, the software can't tell that this is the same screen as another but that just happens to currently be showing a different set of fields. So when you come back to a page and there are different fields on the page than when you initially worked with it in Presto, the software checks its internal files for a match on the characteristics of the panel, doesn't find a match, and concludes it's a different program.

You may also run into situations where Presto does not recognize that a subfile is actually a subfile. This can happen, for example, if the very first time you visit a subfile program the list includes just one record that isn't aligned with the headings. So if you return to the subfile when there are 8 or 10 records showing, Presto may not recognize it as the same screen it's already been to.

Solution

If many of the screens in your applications display some fields only conditionally, you can turn off the environment option to include fields in the screen ID. If you only have a couple of screens where this happens, you can leave that option turned on, but clone your custom changes to create a separate instance for each variation of that screen.

If Presto doesn't seem to be recognizing your subfiles automatically you can manually draw tables around them.

If you aren't certain which alternative is best, please call Technical Support!

Fields and Variable Function Keys in Rows 22 and 23

While you can adjust your auto-detection settings to look for your function keys in any three rows of the screen, rows 22 and 23 are the default.

The default auto detection settings for these rows assume that your subfile list or data fields will normally end on row 21, and that rows 22 and 23 include mostly constant content (e.g. function key text). Row 24 is typically reserved for messages.

In this excerpt of Figure 2-2 you can also see that the auto detection settings expect some variable content in positions 71-80 of row 22 (i.e. the typical locations for More.../Bottom messages):

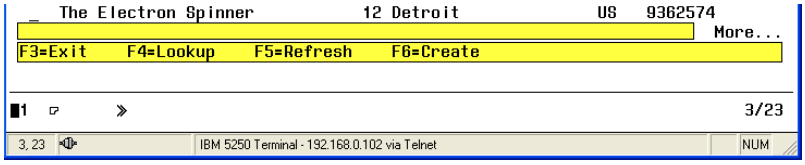


Figure 2-5.

As a result, positions 71-80 are excluded from the mask used to uniquely identify the page.

The effectiveness of the auto-detection settings for rows 22 and 23 depends on:

1. whether these lines can include fields with variable values, and
2. how many of your programs have variable function keys.

Your function keys for a screen may change because some are conditional, or because you have more keys than can be displayed at one time and your programs are set up so users can press F24 to see a different set of keys.

In cases where your green screen programs use F24 to see more keys, a typical tactic in the modernized version of the program (where you have ample screen space) is to create custom HTML that includes *all* your function keys and attach that to every variation of your screens. Then users can see all the function keys at the same time, and you can drop the F24 from the visible UI.

Troubleshooting Page Identity Problems

This section describes a few scenarios that you may run into as you develop your Presto application, but we'll start off with a tip that's useful across the board.

Tip: Be Aware of Presto Internal Key Values

The design tool's title bar displays the key value Presto assigns each page:

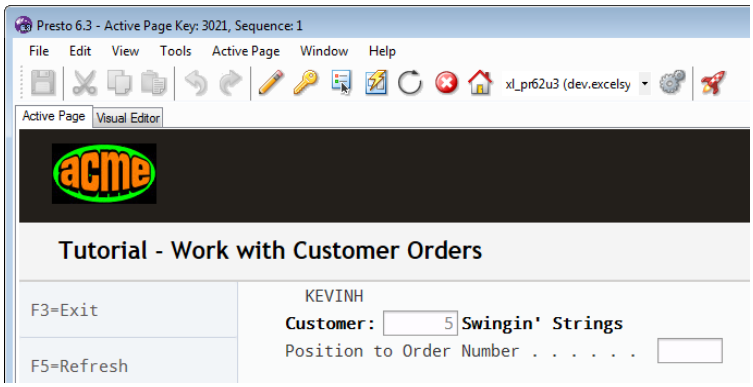


Figure 2-6. The Active Page unique key.

The key value for the page above is 3021, 1. This happens to be a subfile program, and when we roll up and down through the list this value should NOT change. This tells us that customizations will be consistently applied.

When we select display a particular order record, the key changes:

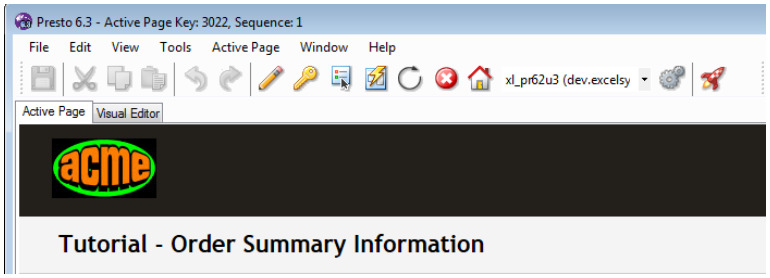


Figure 2-7. Checking another screen in the same program.

While this is just another screen within that same program, the screen has a different number of fields and function keys. As we would expect, the internal key value shown in the title bar is unique (3022, Sequence 1) and different than the subfile page's key value.

If you go to a screen and Presto doesn't apply your custom changes, start your troubleshooting process by comparing the key value each time you leave the screen and come back. If it changes, it means Presto thinks it's looking at a different screen, and it's time for some detective work to find out why!

'Lost' Customizations

Scenario: You install Presto and it's working well out of the box before you start making custom changes. Then you start making changes to the web versions of some of your screens. Unfortunately, in some cases these customizations don't seem to 'stick'. You may initially think you are doing something wrong and add them all over again, only to have them once more disappear the next time you visit the panel.

Explanation: What is most likely happening is that you have a variable within your detection settings. As we have discussed above, there are few ways this can happen:

- If the variable information is all or part of a System Time field, your customizations may *never* appear to stick! Every time you run the program Presto will think it's seeing a new and different page. What can be really confusing is if your detection

settings incorporate just the last digit - your customizations will magically reappear about one time in 10!

- If the variable is a date, your custom features will appear to work well for the rest of the day in which you add them, but when you come back to working with Presto the next day everything is gone.
- If the variable content is a user profile your custom features will work well for you but be gone when a co-worker accesses the application.
- If the variable is something environmental, your changes may appear to be lost when the programs are moved to another system or a testing partition, or run from another location.

The most important thing to internalize is that when customizations appear to have been 'lost' there is usually an explanation!



The sooner you catch these issues the easier they will be to resolve. If you do end up needing to change your page detection settings it's far better to do it early than after you have added custom features to dozens or hundreds of screens.

When you change the page detection settings Presto must rebuild all the internal key values to identify each page. When this is complete you may need to manually re-attach customizations to your pages.

Solution: The section Reattaching Custom Changes, below, describes how you can find and reattach any custom features that have gone AWOL.

Customizations Applied to the Wrong Pages

Scenario: If Presto can't distinguish between two (or more) screens in your system the software may incorrectly apply custom features intended for one screen to other similar ones.

Explanation: This is more likely to happen if:

- You uncheck the option to include fields in the screen identifier.
- You have excluded rows 22 and 23 from your auto detection (perhaps because many of your programs have variable function keys).
- You have many screens with generic titles.

Solution: You can resolve sporadic incidences of this problem by adding global identities. We'll discuss this in the section Using Global Level Identities, on page 124.

Windows with Generic Titles and Function Keys

Scenario: In your applications, custom changes to window programs and panels are particularly prone to being 'lost'.

Explanation: Presto doesn't use the standard auto detection settings to identify windows. Rather, it identifies window screens using the title in the top border and the function keys at the bottom.

If your window borders don't typically include a title, or you have many windows with both generic titles and function keys, you may see custom features designed for one window applied to another.

Solution: You can resolve this by adding a global identity to each window as you customize it. As with uncustomized screens, unmodified windows should always work fine.

Reattaching Custom Changes

Presto allows you to add custom HTML, JavaScript and Transformations to your pages. You can see these options on the right of the Presto Designer:

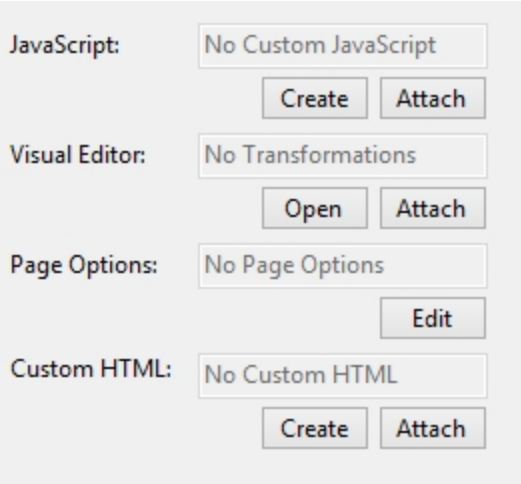


Figure 2-8. Options for attaching custom HTML, JavaScript and Transformations.

The ability to quickly attach customizations to a page makes it easy for you to test various design strategies with your pages, detaching and attaching different alternatives during testing. It also means that if a customization becomes detached from a page unexpectedly, you can easily add it back once you track down how it came to get lost.

When you click any of the Attach buttons above Presto displays a list existing files of that type, which you can preview and select from.

Using Panel Identities to ID Screen Variations

Once you have a broad idea of the auto detection settings that will work best for the *majority* of the screens in your applications, set your environment detection settings accordingly.

It is likely, however, that you will still have a number of screens where the only distinguishing content lies outside the auto-detection areas, or where variable content within the auto-detection areas mistakenly leads Presto to not recognize a screen as one it has already seen.

Consider this example:

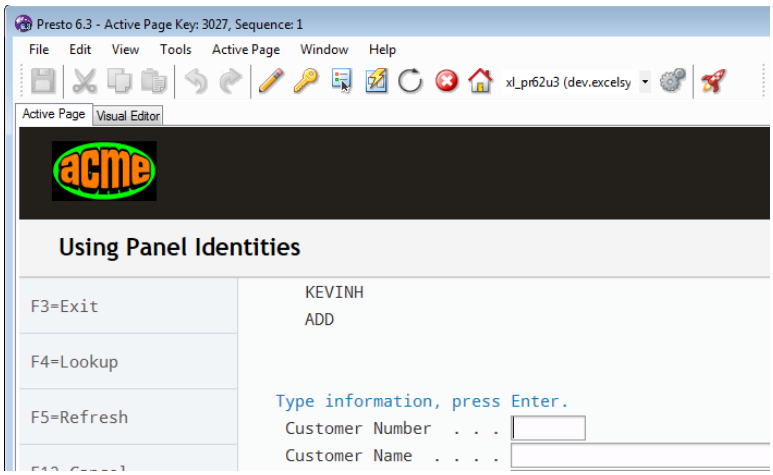


Figure 2-9. A page used to add or update customers.

This is a flat panel from a program used to add and maintain customers. In this case, the only difference between the screens is a 'program mode' field that lies outside the default auto detection areas.

In the title bar, this page is identified as page 3027:

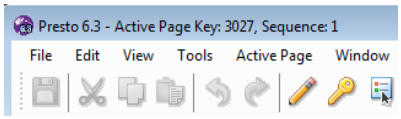


Figure 2-10. Noting the page's unique key value.

This is in the top corner of the window.

When we use the same program to update a record, the screen is nearly identical:

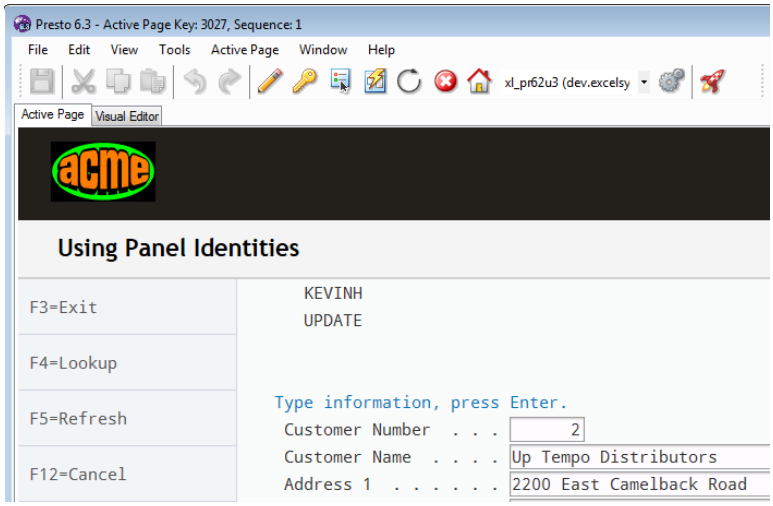


Figure 2-11. Checking the page ID for a different variation of the page.

Notice that even though this is technically a different screen - the program mode is UPDATE - Presto doesn't distinguish between this page and the ADD mode screen.

Now, this may be a good and desirable thing if you plan to add custom features to this page that you would like visible in both ADD and UPDATE modes. But if you want to have unique features for one or the other mode, you need to make sure Presto can tell them apart.

For these particular situations you can define unique Global Identities. Global Identities allow you to take a set of pages that Presto thinks are the one and same, and split them apart so Presto recognizes them as being different.



Note that Global identities are for situations where you can make Presto recognize a page as unique by *adding* screen recognition details that aren't getting picked up by the auto detection settings. If, on the flip side, you have content within the auto detection areas that is variable you need to use a Global Identity.



We recommend you set up as many Global Identities as you will require before adding custom features to the screen, to ensure those features are permanently linked to the right panel. While it's possible to manually attach custom changes to a panel afterwards, it is an avoidable extra step.

Using Global Level Identities

Use global level identities to override Presto's auto detection settings for individual screens.

The Purpose of Global Identities

To help you understand the need for global identities and the role they play, consider the example of one of our customers who is modernizing an application that includes:

- several hundred screens belonging to a legacy tool purchased long ago from a vendor, which can't be changed
- many other programs built in-house over the years, using a consistent, but different, design standard.

The auto detection settings that work well for one set of programs don't work well for the other set, making it challenging to ensure that when pages are customized, the customizations are reliably connected to the right pages.

Global level identities allow you to use one or more screen values to uniquely tag your screens. These identifiers are checked by Presto before the standard detection settings, effectively overriding them.

To illustrate, consider this program in our tutorial application:

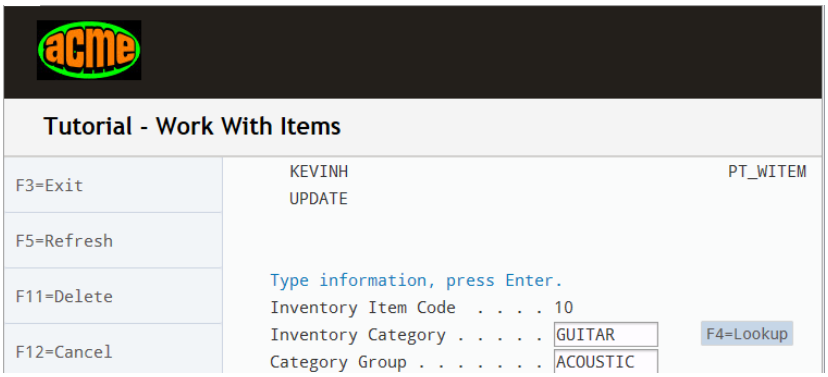


Figure 2-12. Tutorial application, with unique program name in the top right corner.

If every programs and page in our application has a unique name in the top right corner, it's very easy to define effective automatic detection settings.

For this application we could set up detection settings like this:

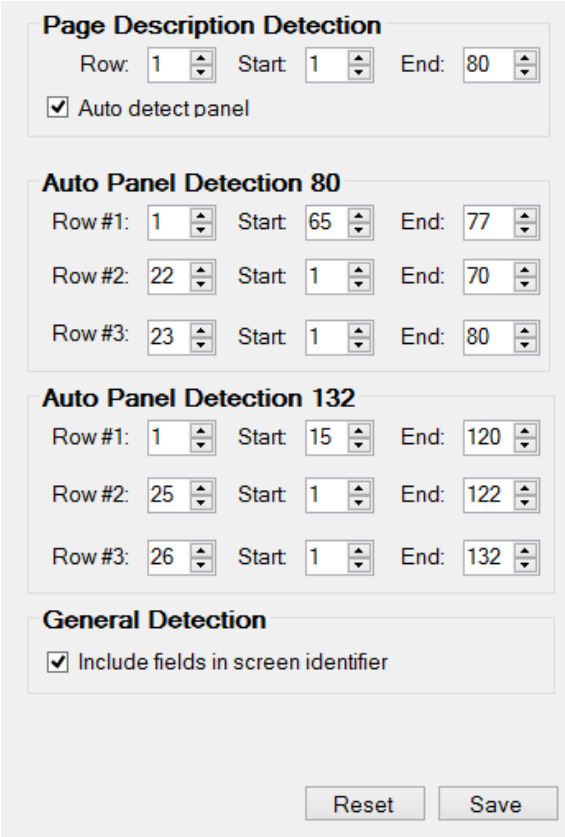


Figure 2-13. Detection settings to capture the unique program and screen name.

Open the live preview of Detection Settings via the Tools>Detection Settings menu option. We can confirm that the unique program name is highlighted:

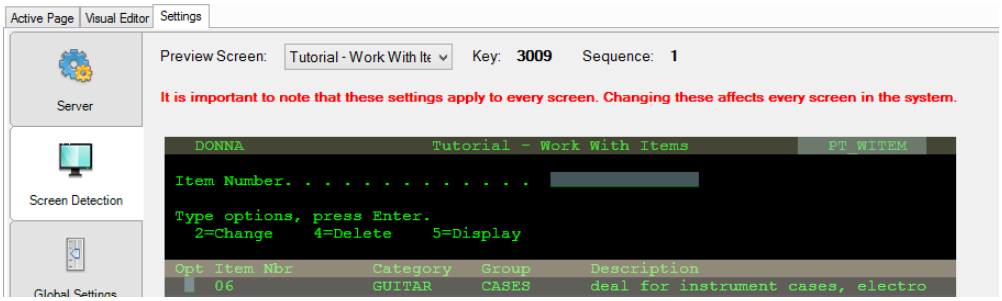


Figure 2-14. Detection settings highlighted.

This situation would be ideal - if it was consistent across every program.

Unfortunately, most environments are not consistent, and attempting to define detection settings that effectively handle every single screen layout variation can be next to impossible.

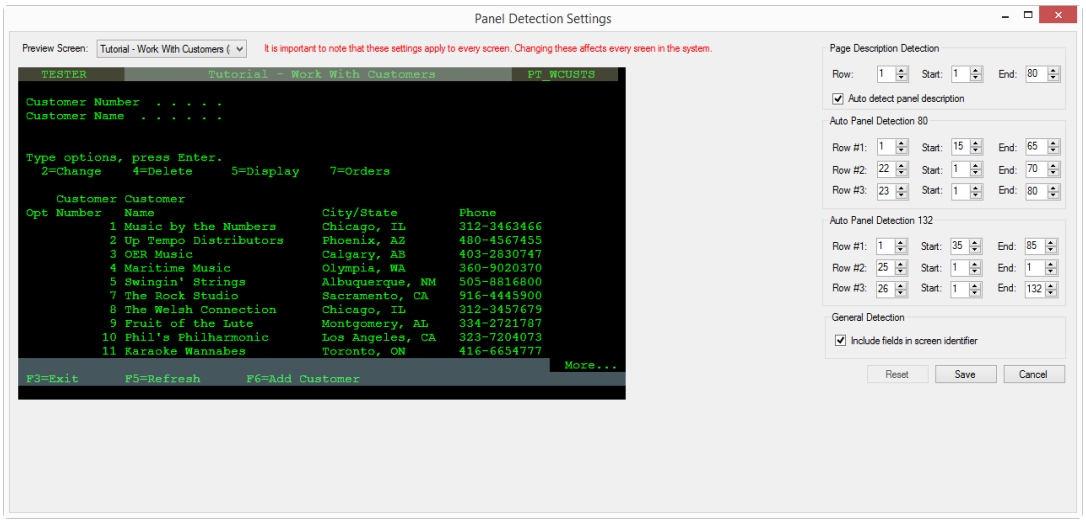


Figure 2-15. Previewing Detection Settings.



It's important to avoid getting into the cycle of pruning back your automatic settings to try to handle every page layout outlier.

The best approach is to use settings that work for the broadest range of programs, then handle outliers with explicit, manually created global level identities.

If you aren't sure what this means, please call Technical Support!

To continue with our example: suppose that our application also includes a number of programs that have both a generic value in the upper right and even a generic program title:

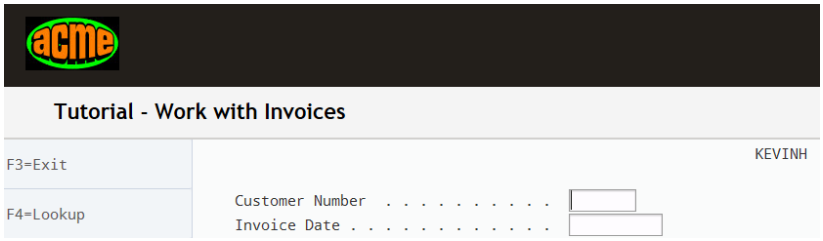


Figure 2-16. A different screen layout, with a non-unique value in the upper right.

Can you see the problem? In this program (and a few others like it) the top right corner includes a user profile. This means that if Presto is looking at just that spot to ID the page, it will be confused not only by finding the same user profile at that location across multiple pages, but other user profiles there when this program is run by different individuals.



If you don't make any custom changes to this screen, then you don't have a problem - Presto will automatically render the web version of the page as it is delivered from the server. The challenge comes when you add custom features to the page - Presto needs to be able to reliably identify this page every time you visit it, and the value in the detection settings area isn't unique.

You can define a global identity on this page to resolve this problem.

Adding a Global Identity

Follow the steps illustrated below to create a unique global identity for a page that doesn't fit the template you've defined in your Detection Settings.

Step 1: Navigate to the page.

Start by navigating to the page in the Presto Designer. In our example, the problematic program is the tutorial's Work with Invoices program:

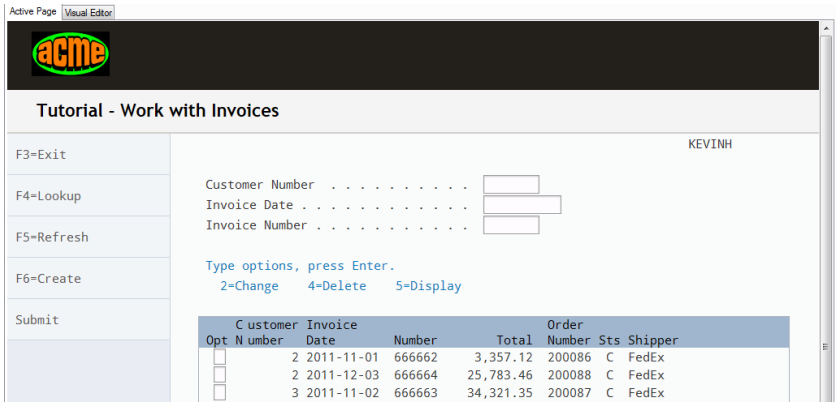


Figure 2-17. Navigating to the page with a non-unique identifier.

As we saw earlier, this program is anomalous in that the program name isn't included in the page at all, and the screen location that in the rest of our application normally includes the program name, here has the User ID.



To be absolutely clear: These differences are only a problem when you are adding custom features to the screen. The page, of itself, poses no challenge to Presto if you are just rendering it out of the box, using your default skin and global custom JavaScript enhancements.

However, if you change the page properties, add a custom page-level script, add visual editor transformations or change the page HTML, Presto needs to be able to always uniquely identify this page in order to reliably attach those changes to this page, and this page alone.

Step 2: Go to the Global Identities.

Click the Identities button:

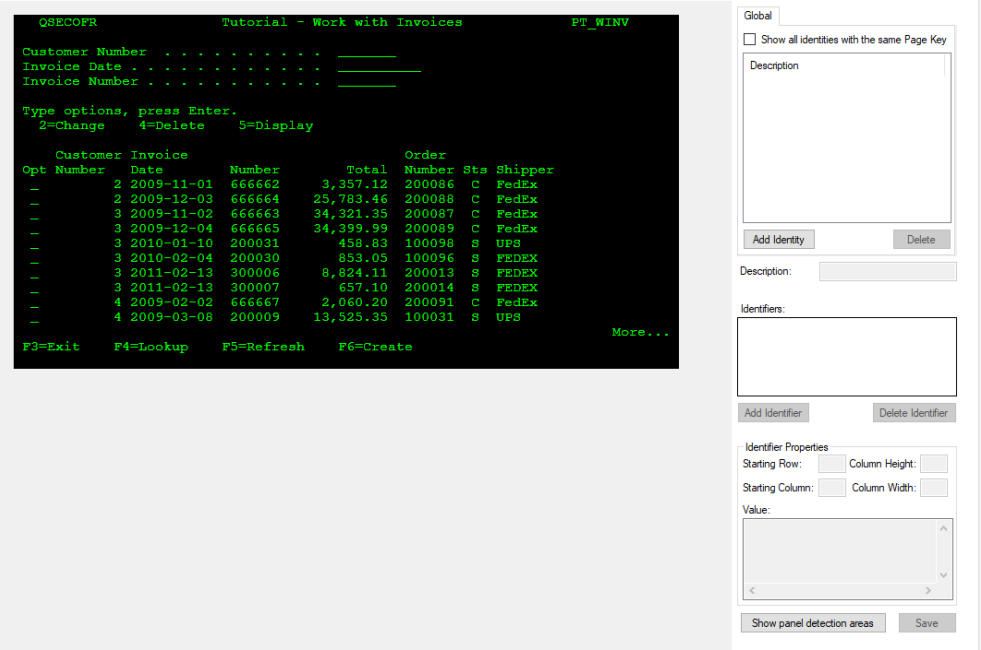


Figure 2-18. Preparing to work with Global Level Identities.

The left side of the screen shows a preview of our application.

Step 3: Analyze the page to find unique, consistent characteristics.

What we need to do know is identify some constants on this page that will be there every time this page is accessed, and distinguish it from all over pages in our application. The most fertile ground for this exercise is usually the header area of the page, though unique patterns of function keys or field headings can also be useful.

Let's look at this page a little more closely:

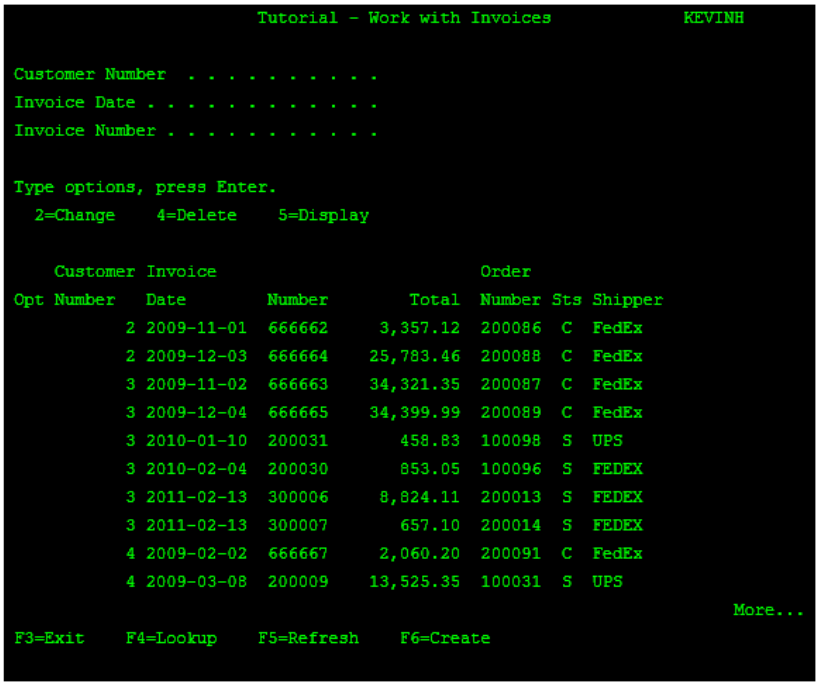


Figure 2-19. Identifying unique characteristics of the screen.

There are a couple of strategies you can pursue:

- You can pick a single, unique combination of words that is highly unlikely to appear in any other page. For example, perhaps no other program will have exactly the same title, the same position-to fields, or identical column headings.
- Or you can add multiple identifiers, to make absolutely sure that no other program will match.

There are a few things you need to keep in mind:

- Will this page change as the user rolls through the list?
- Could this program have a somewhat different appearance if it was called by another user, or passed different parameters?
- Does this program include other conditional content?

The answers to these questions will also guide your strategy in defining your global identify.

Step 4: Add the Identity.

In this case we will play it safe by building our page identity using both the program title and a couple of heading text entries. As mentioned above, we could consider using the function keys, but in this case they seem quite generic.

Click Add to get started. When you do this, a few things happen:

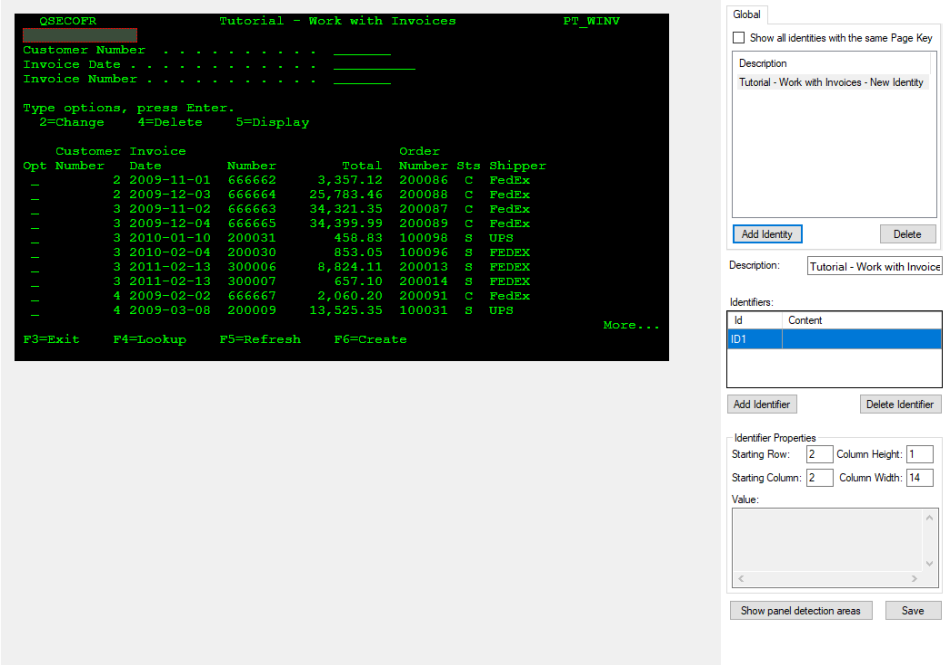


Figure 2-20. Adding the first identifier.

In this illustration we have just clicked the Add Identity button (click past the warning message). When we did this, Presto dropped a preliminary identifier trap into the preview, and has assigned it a default ID (ID1) and description (Tutorial - Work with Invoices).



If you want to follow this example, this is the Work with Invoices program from our tutorial application.

Drag the trap from its default location up to line 1, and extend it to include the program title:

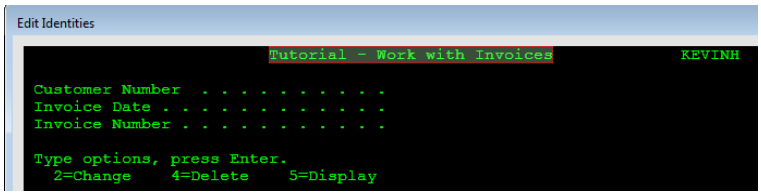


Figure 2-21. Redefining ID1.

When we make this change, Presto updates the identifier properties:

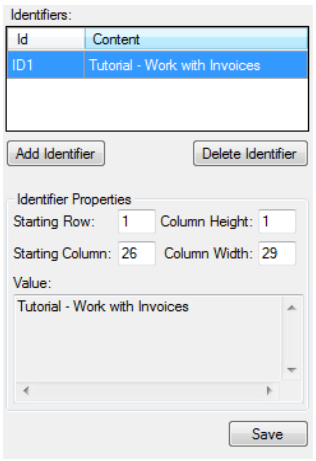


Figure 2-22. Confirming the updated identifier properties.

Step 5: Save your changes.

At this point you can save your changes.

Step 6: Add a second identifier (optional).

As we said above, to be perfectly safe we are going to add a second identifier to this page, to make absolutely sure that this page can't be confused with any other page in our application. Because our subfile always includes the same fields, the subfile headings are consistent (with a folding subfile this might not be the case), so when we click the Add Identifier button to create a second identifier, we'll drag it over to the column headings on the right:

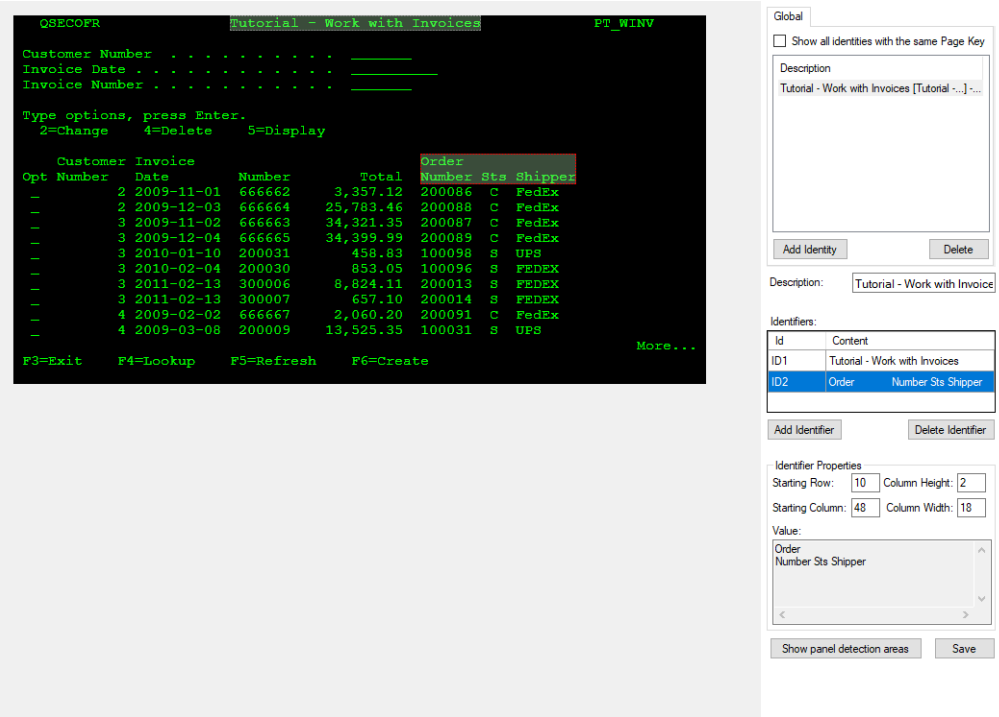


Figure 2-23. Adding a second identifier.

In this illustration we have

1. Clicked the Add Identifiers button in the Identifiers section.
2. Dragged the trap over to the column headings, and resized it to include both lines.

Then clicked Save.

And you're done!



To help you avoid page collisions Presto checks that each identifier is unique across all identities. In other words, if you create an identifier that is used in any other global identity, Presto will issue a warning message alerting you, and recommend you add further identifiers to ensure the identity is unique.

As a result of these changes, before attempting to match this page using the automatic detection settings, Presto will check the list of global identities you have manually defined to see whether you have created one for this page. The User ID in the upper right is ignored, and all that matters is that the program title and column headings match the values we defined.

Step 7: Confirm.

There is a quick way to tell if a page is using a global identity rather than the standard environment level detection settings. When you navigate to the page, the Active Page Key will be zero:

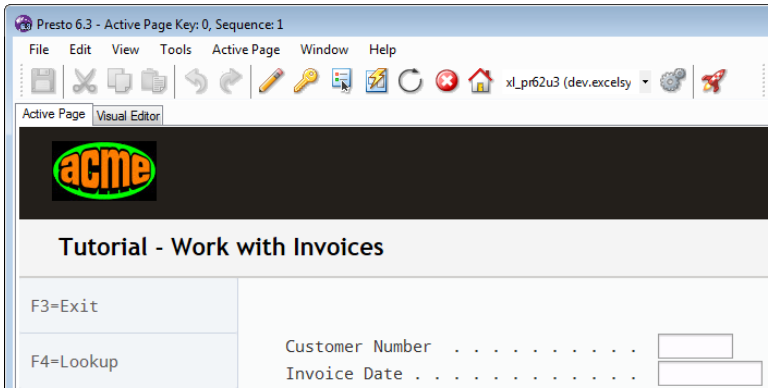


Figure 2-24. Checking the Active Page Key.

The Active Page Key value for this program is 0, with a sequence number of 1. This is what we expect:

- Global identities all have an Active Page Key of zero, and
- this is the first global identity we have defined in this environment.

If the Active Page Key has a non-zero value, Presto is using its standard detection tools to ID the page.

Pages that Match More Than One Global Identity

As you are defining global identities you may occasionally see this error message:

```
Identity is not unique. A match was found in Identity: Tutorial - Work with Customers.
To resolve, add another identifier or change the existing one(s) on either of the pages.
```

This message is advising you that another global page identity matches this page, and that you will have to take additional steps to avoid a page collision.

How are Identities Prioritized?

Let's imagine you have defined 20 global identities for your application, each with a single identifier (an 'identifier' is a unique text string, named ID1, ID2... that makes up the identity). Each time the user navigates to a new page Presto very quickly rips through those identities in sequential order, and uses first matching record in delivering the page to the user.

However, if you define identities using more than one identifier, Presto gives these priority, checking against them first, on the assumption that they are the most specific and should therefore be given higher precedence.

Taking this fact into account, when you get the warning message above, you have two options:

- Change the identifier in your original identity so that it is more unique to that original page, and does not also match the page you're currently working with.
- Add a second (or more) identifier to this page's identity definition.

Adding two or more identifiers to a page directs Presto to give those identities priority in looking for matches.



If your site has many custom pages (hundreds or more) and you are using global identities as your primary means of identifying pages, you can save a few CPU cycles by adding extra identifiers to your most frequently accessed pages, forcing them to the front of the queue when Presto is comparing pages to identities.

Disable Page Identities

This option hides the page identities tab. It is available by pressing Ctrl+F11. It is disabled by default.

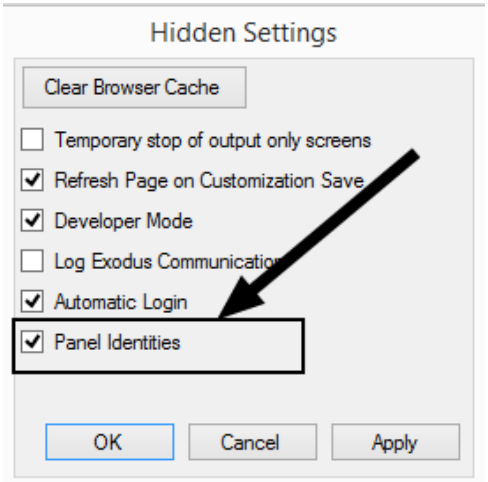


Figure 2-25. Panel Identities Option (Ctrl+F11)

If you are already using Page-level identities, and that approach is working fine for you, you may leave the Page-level identities tab enabled.

For a new site and new customizations, we recommend that you use global identities. In that case, you may want to disable this option and only view the Global Identities tab.

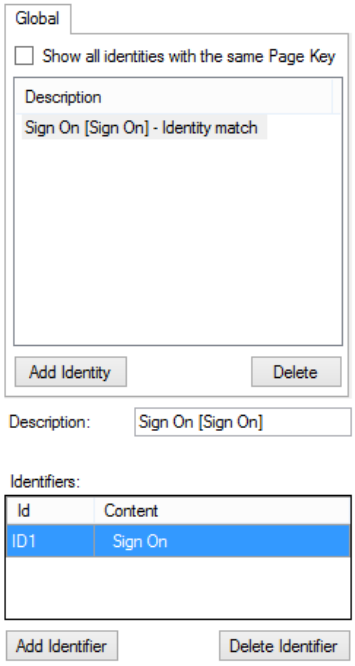


Figure 2-26. Panel Identites with Page-level Identites disabled.

A Final Word of Caution

Presto is designed to use your environment's Detection Settings to efficiently and automatically identify the hundreds or thousands of pages in your application.

When you define global identities you are effectively stepping outside those efficiencies to hard-code a list of unique 'matches' that Presto must check each and every time any user navigates to a page. If you use global identities to handle exceptional cases there will be no discernible performance impact. At the other extreme, if you use global identities as your primary mechanism for identifying hundreds of pages, and you have hundreds of users accessing those pages all day long, you may experience some performance degradation.

If in doubt, get in touch with Technical Support!

Chapter 7 - Customizing with the Page Properties

You can use the Presto Visual Editor tools to customize individual items on a page as well as features affecting the entire page. To work with your Page Properties you can click the Edit option at the right of the Active Page tab, or as we have done below, select the Page Properties in the Visual Editor:

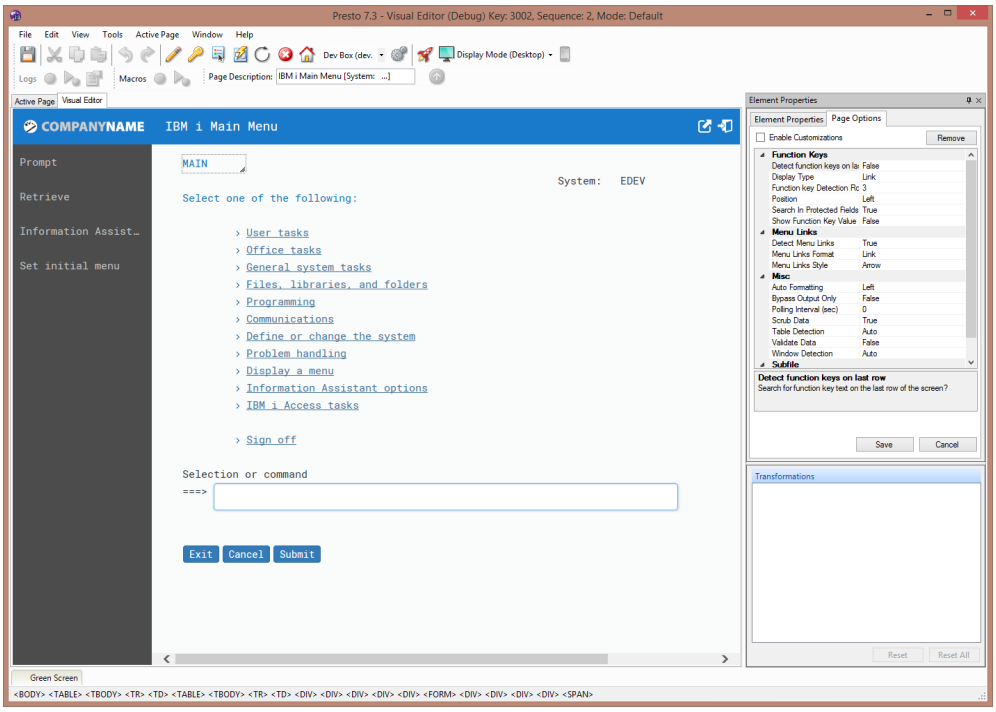


Figure 2-1. Custom options enabled in the Designer.

In the sections below we'll start by looking at the simpler options and then dedicate more time to the function key, table and window creation tools.

Working with Page Properties

To work with the page properties click 'Enable Page Property Customizations'. This enables the options and features described in this chapter.

Saving Changes

When you check this option you must click the Save button to retain your changes:

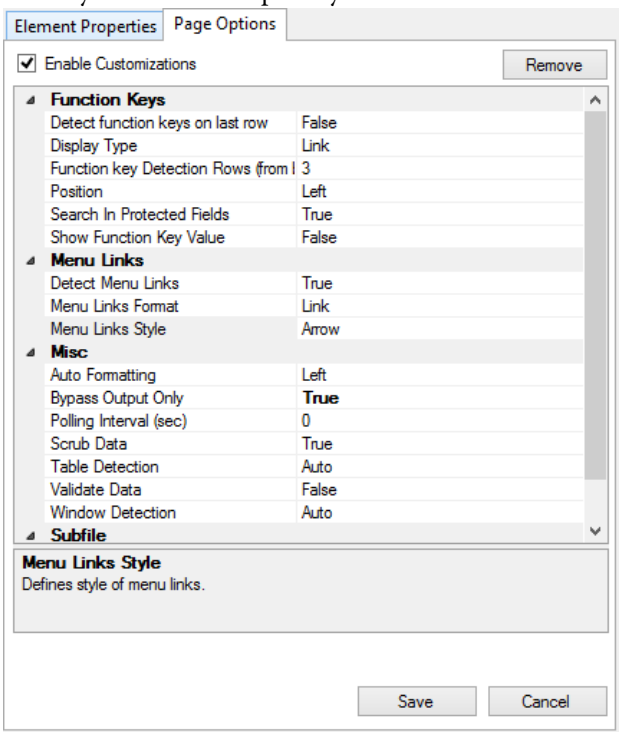


Figure 2-2. Preparing to save changes to Page Properties.

Click the Save button in the bottom to save your changes. The Visual Editor tab will show an asterisk any time you have unsaved changes.

Temporarily Disabling Changes

You can temporarily disable or remove your changes by unchecking the 'Enable Page Property Customizations' option and clicking Save.

Removing Changes

To permanently remove your changes click the 'Remove' button, then save. Removing your changes reverts to the environment level settings.

Page Properties

The remainder of this chapter discusses each of the page properties. Note that some options are also available as Global Settings. They are discussed in the section “Global Settings” on page 42.

Function Key Type & Text Options

Here is an illustration of a typical list-style program with the function keys as buttons on the left, and including the ‘Fxx=’ text:

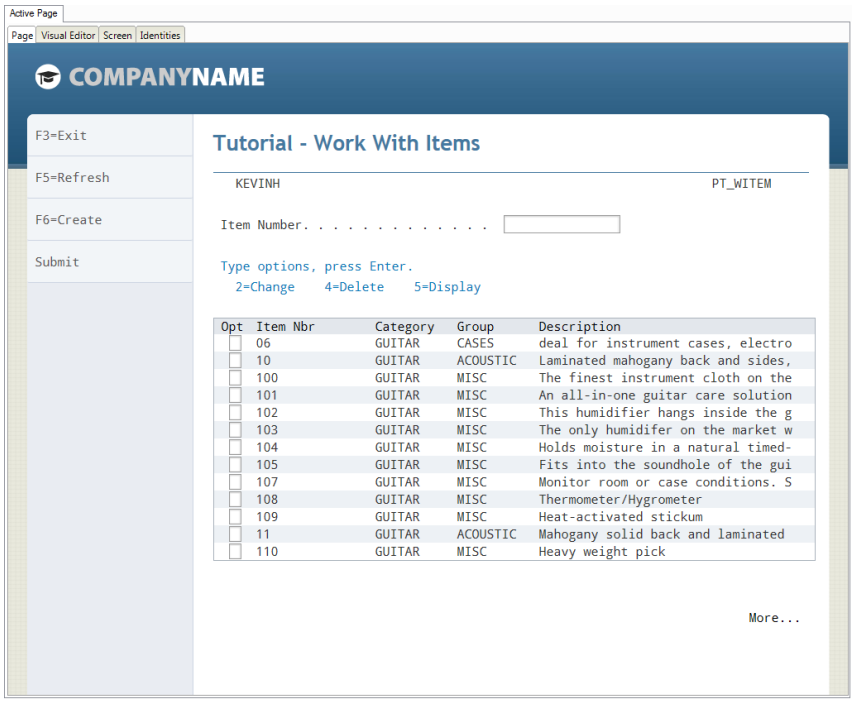


Figure 2-3. A typical list-style program, as rendered by presto for a web browser.

If you want to change the way the function keys are displayed on a particular panel go to the Visual Editor and select the Page Properties tab. Then click the option to Enable Page Property Customizations. (Figure 2-2).

Detect function keys on last row

Presto by default expects that your panels will use line 24 as a message subfile. However, if your program does NOT have a message subfile on that line, any fields or constant text strings in line 24 of your program will not be shown.

Function Key Display Types

The supported function key display types include:

- **Text.** Use this option to not perform any conversion to buttons, leaving the function keys as non-clickable text entries on your pages.
- **Button.** When you select this option your function keys are rendered as clickable buttons.
- **Links.** Use this option to display your function keys as text links.

When you select Text your users can only use function keys by pressing them on their keyboards.

Position

Supported position values include Left, Right and Bottom. Figure 2-3 illustrates the Left option.

Search in Protected Fields

If your convention for function key text isn't distinctive or you have odd data in your files, it's possible that Presto may believe it has found function keys embedded in your field values, and display them as buttons.

This option is available at the environment level (and discussed on page 48), but if it's happened just in a particular program or two, you can turn off the 'Find Function Keys in Protected Fields' option and try reloading the page.

Show Function Key Value

When you check this option Presto displays function key text as 'F3=Exit' (showing the Fxx=). If you do not wish to include the 'Fxx=' portion of this string, uncheck this option to show only the function key action.

Detect Menu Links

You can toggle Presto's automatic feature to convert menu options to hyperlinks at both the environment and page level. This means you can set the most appropriate default for your environment but also override it for individual menus or pages that need to be treated differently. For example, Presto's automatic menu detection could theoretically be confused by certain styles of numbered lists that aren't actually menus. In these cases you could leave on the Menu Link detection enabled at the environment level but turn it off for those particular screens.

Menu Links Format

Use this option to set the default appearance of your menu links. The options include:

- **Link.** Use this option to render the menu options as links.
- **Button.** Select this option to render the menu options as clickable buttons.

Menu Links Style

This option is designed to be used in conjunction with link-style menu links. The options include:

- **None.** Use this option to not perform any styling on the menu links. They will work as clickable links.
- **Arrows.** Use this option to render an arrow beside the menu link.

Adjust Last Row

This option allows you to adjust row 24, moving it up, to remove the empty rows created after repositioning the function keys.

Here is an illustration of the gap created when function keys are re-positioned to the left sidebar:



Figure 2-4. After moving function keys to the left sidebar, rows 22 and 23 are empty.

Here is an illustration of setting the Adjust Last Row to True.

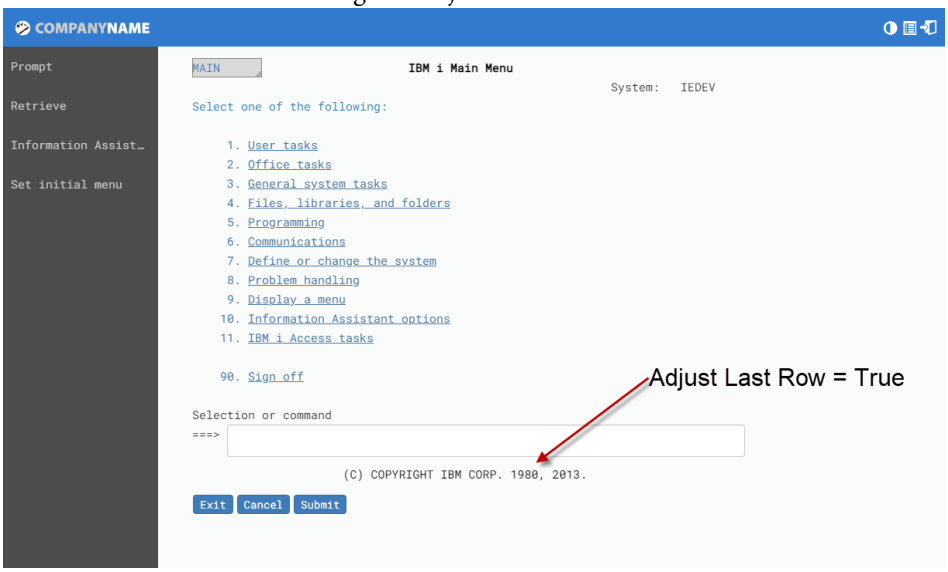


Figure 2-5. Adjusting row 24 up to row 22/23 to remove the empty rows.

Note that this option can be set globally. See “AdjustLastRow” on page 61 of Chapter 2.

Auto Formatting

The automatic formatting option allows you to replace all text strings formatted like this:

```
Text Description . . . . . .
```

with this alternate style:

```
Text Description:
```

This feature is also available at the environment level.



If the patterns of dots or other characters on your screen don't match Presto's out of the box settings, you can also define regular expressions to control one or more styles of dots to be removed from the screen.

You can enable this feature via the <autoformatpatterns> option in config.bws (Tools>Advanced Configuration). Contact Technical Support if you need help with this.

Left/Right Alignment

When you enable Auto Formatting you must choose your text alignment preference. You can set whether the affected text is aligned left, against the edge of the panel, or right, adjacent to the field.

Bypass Output Only Screen

This setting is also available as an environment level option.

This option works in conjunction with 'Temporary stop output only screens' IDE option, available by pressing Ctrl+F11. Use this feature to allow you to customize screens that normally pop up only for a second before automatically advancing.

Here is how you can use these options:

1. When this option is unchecked and the 'Temporary stop output only screens' option is checked, Presto pauses for a few seconds on output only screens (allowing you enough time to switch over to the Visual Editor to customize them). This is for editing output screens.
2. When this option is unchecked and the 'Temporary stop output only screens' option is also unchecked, Presto displays the output screen only for a moment. This is for displaying output screens.
3. When this option is checked, the 'Temporary stop output only screens' option is not relevant.

You can set this option at the environment level (if you are interested in working with any output only screen) or at the page level (when you want to work with a specific screen).

Polling Interval (Seconds)

Use this option to enable polling and set a polling interval for this particular screen. This option is discussed in more detail in the section "Activating Polling Support" on page 408, Advanced Technical Topics & Troubleshooting.

Detect Subfile Options

Set this option to True to have Presto attempt to automatically detect subfile option text above your lists, and make those options available to the user when they mouse over the options field.

Here is an illustration of this feature at work:

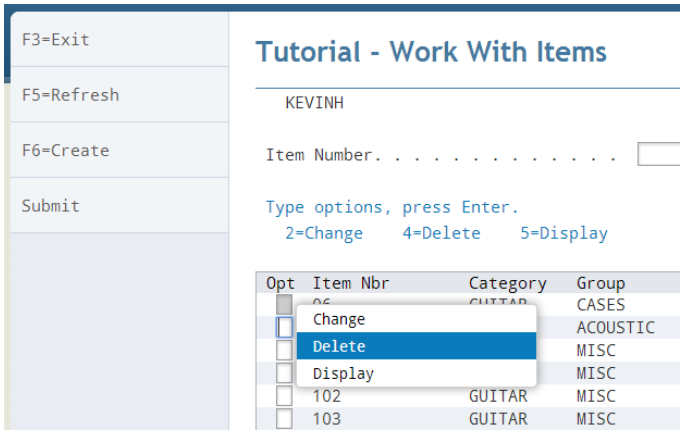


Figure 2-6. Three subfile options detected, and presented, on a list style program.

You can toggle Presto's subfile option detection at both the environment and page levels. This means you can set the most appropriate default for your environment but also override it for individual programs as required.

Hide Subfile Options

Set this option to have Presto hide the subfile options about your subfile.

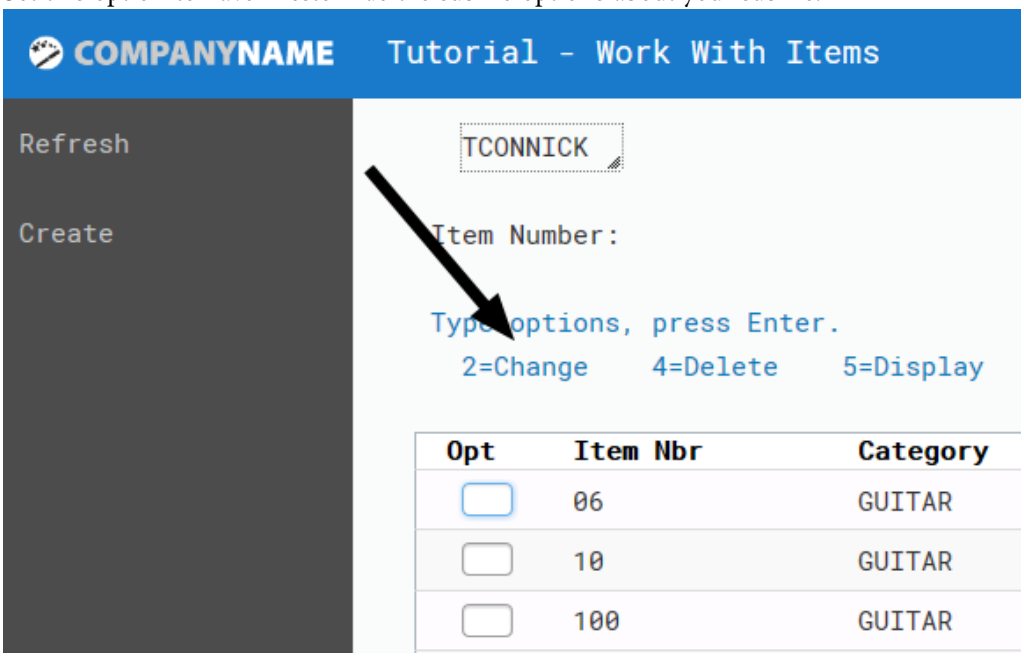


Figure 2-7. Three subfile options presented about a list-style program.

You can toggle the Hide Subfile option at both the environment (Tools>Settings) and page-level options.

Table Detection and Custom Tables

Presto is set to convert your subfiles to tables by default. Below is an illustration of a list-style program where a table has been automatically built around the subfile portion of the screen:

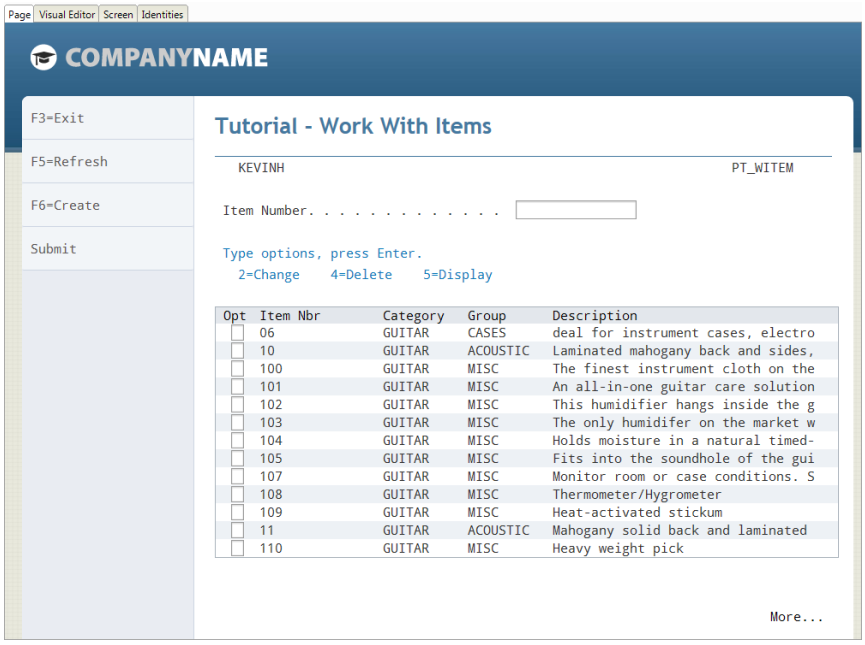


Figure 2-8. A subfile page built using the automatic table creation mode.

Presto automatically scans your pages to see if they contain subfiles, and where found, renders them as HTML tables. The most visible benefit from this is the alternating row colors you can see in the illustration above.

If, in this same program, we override the automatic setting, so Presto does NOT build a table on that panel, the list looks like this:

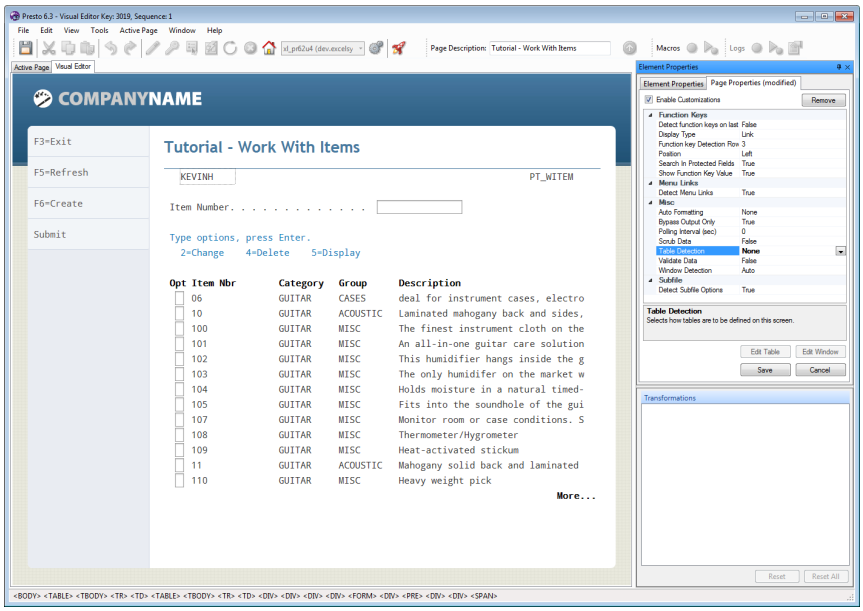


Figure 2-9. A subfile list, with the automatic table creation mode turned off.

Note that in this image the subfile uses the default font and spacing from the skin, and the subfile rows aren't grouped together into a table.

In some cases though, inconsistencies in your subfile contents or structure make it hard for Presto to recognize a subfile as such. In these cases you can manually add a table, as shown in the following sections.

Creating and Working with Custom Tables

While Presto automatically converts subfiles to tables, you can also manually create custom tables on any panel.

To create or work with a custom table go the Visual Editor, then click the Page Properties tab. Enable page property customizations, then select the Custom table detection option, as we are about to do here:

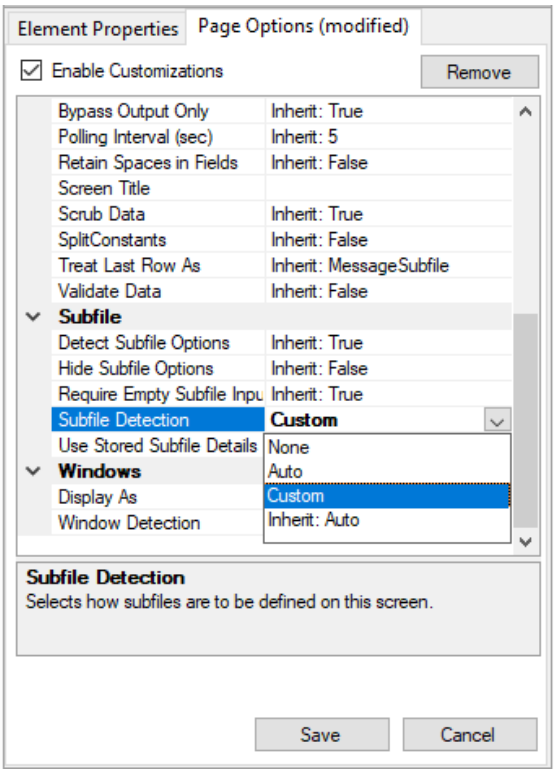


Figure 2-10. Preparing to edit a custom table.

When you select the Custom option, Presto jumps to the Visual Editor tab, illustrated next.

This is the table editor:

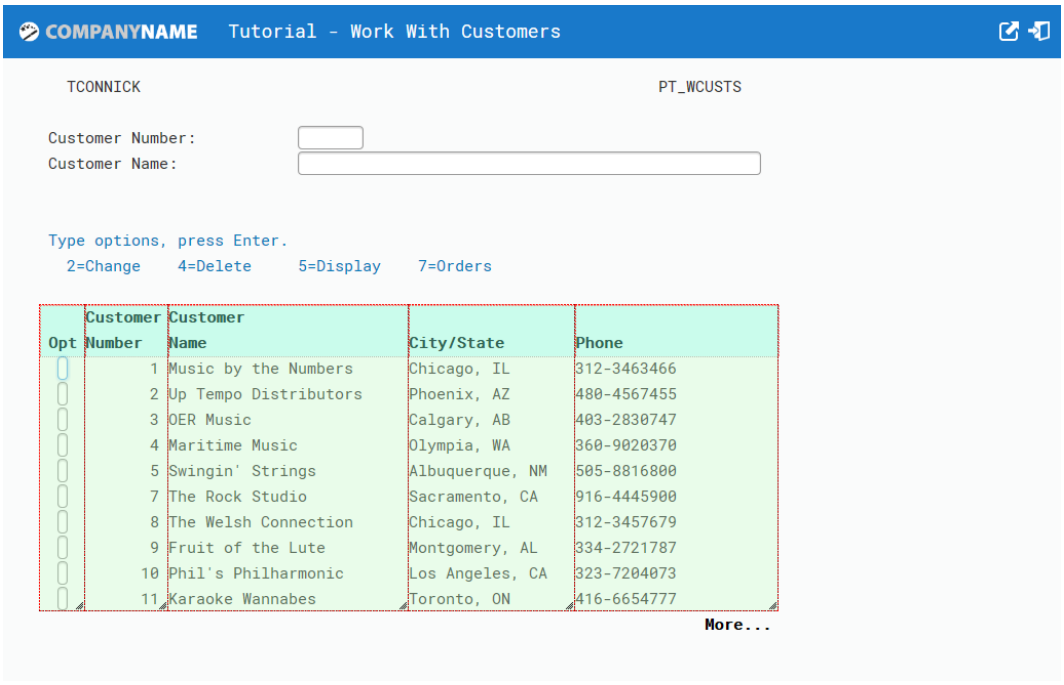


Figure 2-11. Working on a custom table.

In the illustration above, Presto has flipped over to the view of your original screen and presented the custom table creation tools. These tools include the Table Properties and the table

model. You can see that the initial table model has been built over the subfile, because it was recognized as a pattern.

When you create a custom table you need to do a few things:

1. Correctly identify the outer boundaries of the table area.
2. Add the correct number of table columns.
3. Decide whether you need headings and optionally add them.
4. And, where relevant, indicate that the table is built over a subfile.

The text in the table header is shown with different formatting options (which comes from the style sheet associated with your skin).

The next sections describe the process for building tables, but first, here is a description of the available options:

The Table Editor Right-Click Menu

There is a menu that pops up when you right-click anywhere in the table model. From here you can edit these options:

- **Apply Changes**. When you're done, you can click this option to save and apply your changes.
- **Close Table Editor**.
- **Add Header**. Click this option to add a header to your table. The area marked by the header will be formatted differently (as set in your skin settings).
- **Remove Header**. Click this option to remove the header from your table.
- **Add Column**. Click this option to add a new column.
- **Remove Column**. Click this option to delete the right-most column.

When you're done, you can click the Save button to apply your changes.

The next sections describe the process of building a custom table.

Setting the Table Perimeter

The first step in building a custom table is to set the boundaries of your table model to correctly identify the outside boundary of the table. You can do this by dragging the borders of the table that Presto dropped onto the screen by default.

Setting the Table Header

To add header rows to your table right-click the table and select the 'Add Header' option. When you do this, you'll see that Presto changes the color of the first row of the table. To make the header taller, drag the bottom edge of the section down for as many rows as you need.

Table header rows are formatted differently from body rows based on your skin settings.

Adding Table Columns

To add new columns, right click the table and select the 'Add Column' option.

Once you add columns you can adjust column dimensions by dragging the edges right or left to where you need them.

Reviewing Your Changes

When you are finished making changes to your custom table you can right click the table and select the 'Apply Changes' option. If the table isn't quite perfect you can right-click and select the Modify Table option to redisplay the table editor.

Working with Windows

Presto automatically attempts to build HTML windows over screens that it recognizes as windows. When this option is enabled, a typical window presented by Presto might look like this:

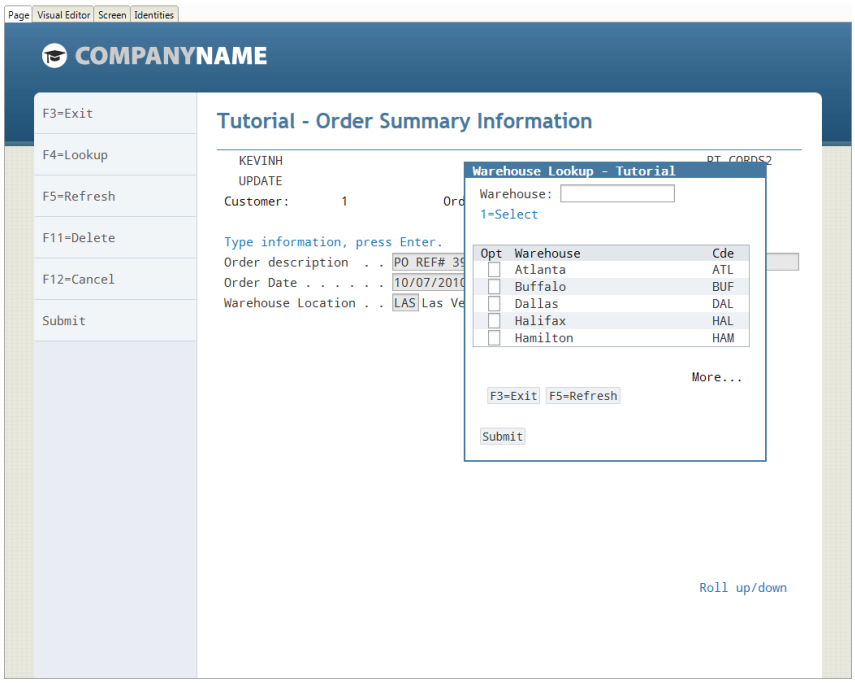


Figure 2-12. An automatically built window in Presto.

Presto displays the window with a simple border around it and the heading has a blue background. Beyond that, the content will normally look much like any other panel (depending on your IBM i design standards, of course).

Creating Custom Windows

In most cases where you call an actual IBM i program configured as a window, Presto won't have any trouble recognizing it and will format it appropriately. One issue you may find is that because windows are often designed to be quite compact, Presto may sometimes struggle to automatically distinguish the separation between the list, list headings, and subfile options. But other than this wrinkle, the auto-detection for windows is the same as the process that builds your tables, and the steps for customizing windows are virtually identical with those outlined in the section above for working with custom tables.

To create or edit a custom window, go to the Visual Editor, then right-click anywhere on your page. Then select the Modify Window option as shown in here:

This is Window Editor:

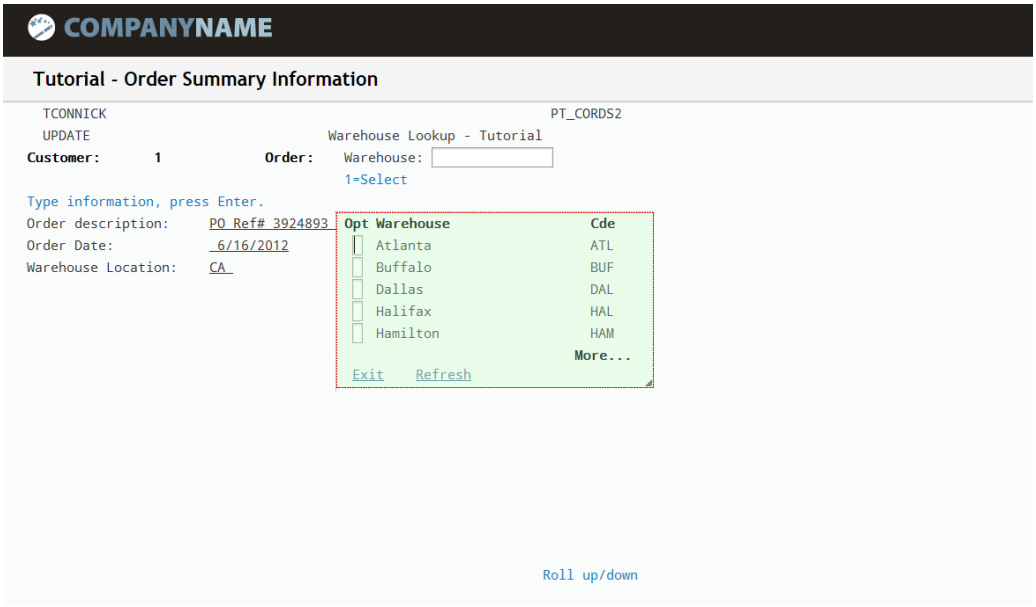


Figure 2-13. Working on a custom window.

You can build the custom window by dragging the borders of the window that Presto created on the screen by default.

When you are finished your changes, you can right click any area within the table and select the 'Apply Changes' option. If you wish to make further changes, you can right-click while in the Visual Editor and select Modify Window.

Working with Window Options

In Page Options, we've updated the Windows options:



Figure 2-14. Windows Options
You have the option to display windows as either a window or a panel:

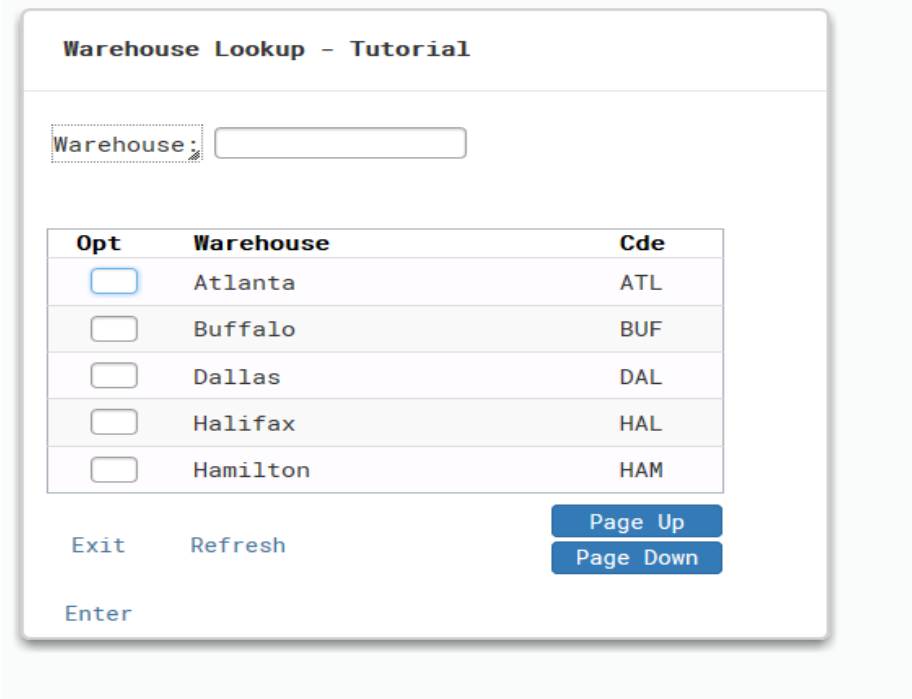


Figure 2-15. Window displayed as a window

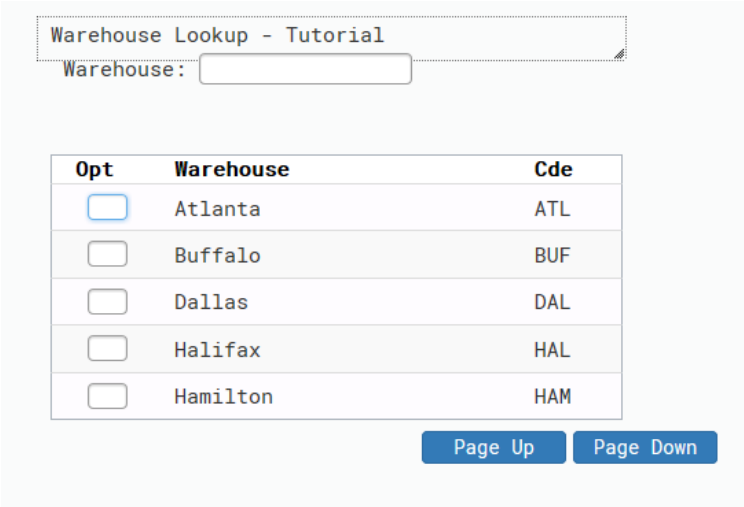


Figure 2-16. Window displayed as a Panel

Chapter 8 - Global Rules for Page Options

Presto provides a system which allows the user to create a set of general rules for identifying screens and customizing the page properties of those screens. It is intended for applying the same custom page options to groups of similar screens, rather than the existing systems which are designed for uniquely identifying screens.

For example, you may want to update all Menu screens to disable table detection or set the menu links to display as buttons. Assuming your Menu screens share certain identifiable characteristics, you should be able to create a rule that will match them all and apply the options changes automatically.

Page Options Inheritance was added alongside Global Rules in order to support the ability to layer the page options config, essentially creating a stack of configurations with the environment settings at the bottom, custom page-level changes on top, and any Global Rules that apply to the screen in the middle. Any config further up the stack can have any options set to “Inherit” from further down in the stack, making it possible to only change values that need to be customized while still allowing the environment config to be adjusted and have it apply to all screens which inherit those values.

The following scenarios explains the hierarchy of the system:

Scenario 1a: 'Inherit' defaults

	Defaults	Page Options	At Run Time
Menulinks	Auto	Inherit	Auto
Subfile Table	Auto	Custom	Custom
Function Keys	Left	Inherit	Left

Scenario 1b: 'Inherit' default - changed environment settings

	Defaults	Page Options	At Run Time
Menulinks	Auto	Inherit	Auto
Subfile Table	Auto	Custom	Custom
Function Keys	Right	Inherit	Right

Scenario 2: Global Rule Options

	Defaults	Global Rule Options	Page Options	At Run Time
Menulinks	Auto	None	Inherit	None
Subfile Table	Auto	Auto	Custom	Custom
Function Keys	Left	Inherit	Inherit	Left

Scenario 3: Multiple Global Rule Options

	Defaults	Global Rule Options #2	Global Rule Options #1	Page Options	At Run Time
Menulinks	Auto	None	Auto	Inherit	Auto
Subfile Table	Auto	Auto	None	Custom	Custom
Function Keys	Left	Right	Inherit	Inherit	Right

The effect of the system is that a Global Rule can override some or all of the base environment configuration for Page Options. Multiple rules can apply to the same screen, potentially overriding the same or different options. Page-level customizations can then override some or all of the previously applied options.

Global Rules for Page Options Manager

Accessing the Global Rules for Page Options Manager

To view the Global Rules, select Global Rules for Page Options from the Tools menu.

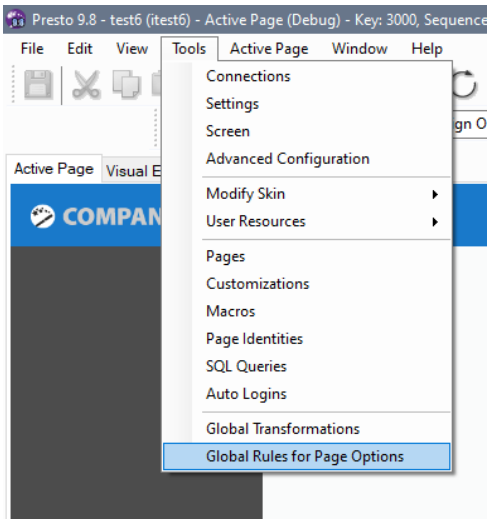


Figure 2-1. Viewing Global Rules for Page Options.

Managing Global Rules for Page Options Entries

The Global Rules for Page Options manager lists all existing Global Rules. You can create or delete any entry using the left list view. The rules can be reordered by clicking and dragging them or by selecting one and then clicking the up and down arrows directly to the right of the pane. You can double-click a rule or click the checkbox to enable or disable the rule.

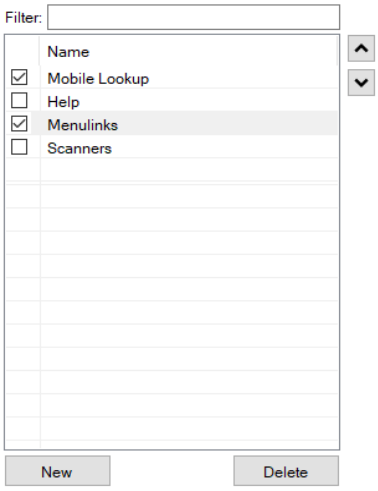


Figure 2-2. Viewing Global Rules for Page Options entries.

New Global Rules and any Rules which have been modified will have a * beside their name in the left pane.

Working with Global Rules for Page Options

The Global Rules for Page Options manager allows you to enter the matching criteria and associated option to perform. You can see these options on the right side of the Global Rules for Page Options manager. This area of the utility is broken up into several sections:

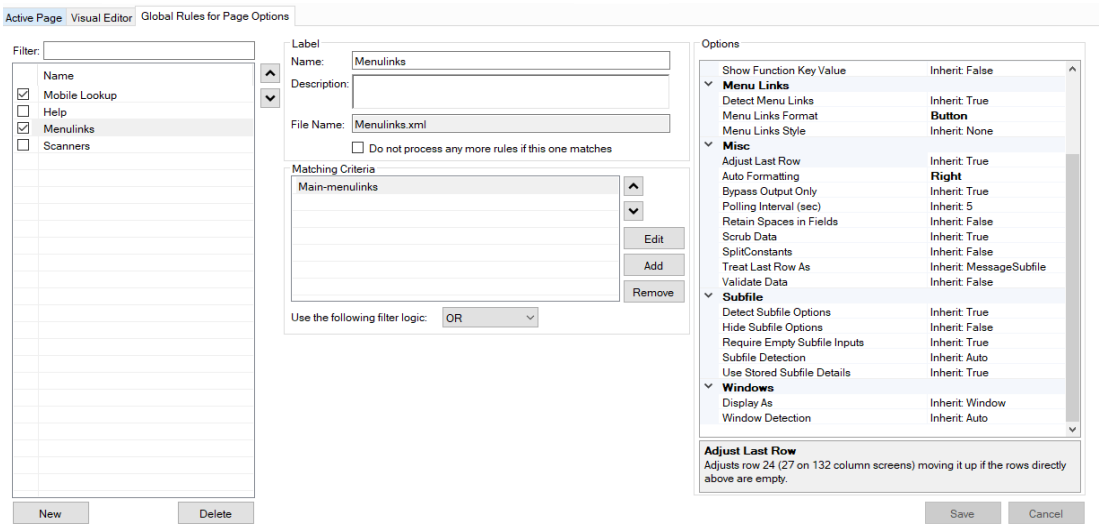


Figure 2-3. Global Rules for Page Options tab.

Label

- **Name.** The name for the global rule. Names must be made up of alpha-numeric characters and must be unique. Spaces are allowed.
- **Description.** The Description field is not mandatory but allows you to provide a more detailed description of the rule.
- **File Name.** The File Name field is read only and will be populated on Save. The file name will match the Name field but with spaces replaced with underscores (_).
- **Do not process...** The "Do not process any more rules if this one matches" checkbox provides a mechanism to short-circuit the Global Rules engine when a rule matches a screen. If a rule with this option checked matches a screen, no further Global Rules will be run against the screen. Make sure that if you have multiple rules that *should* apply to a screen, only the last one in the list has this option checked.

Matching Criteria

The list of Matching Criteria in the middle works very much like the Global Rules list on the left. You can reorder the list as described for rules.

- **Add.**Adds a new Matching Criterion.
- **Edit.** Edit the Matching Criterion selected in the list.
- **Filter Logic.** The Filter Logic drop down box allows you to choose the logic for the rule. At the moment, the options are AND or OR. These options apply the corresponding logic operator to the list of Matching Criteria. Bear in mind that matching of Criteria will short-circuit based on the operator being applied. For example, a set of Matching Criteria with AND logic will stop matching the criteria after the first *false* return.

Options

The right-most pane of the Global Rules Tab displays the Page Options that will be applied to any screen that the current Global Rule matches. All properties default to an **Inherit** state, and will display the page option set in Global Settings.

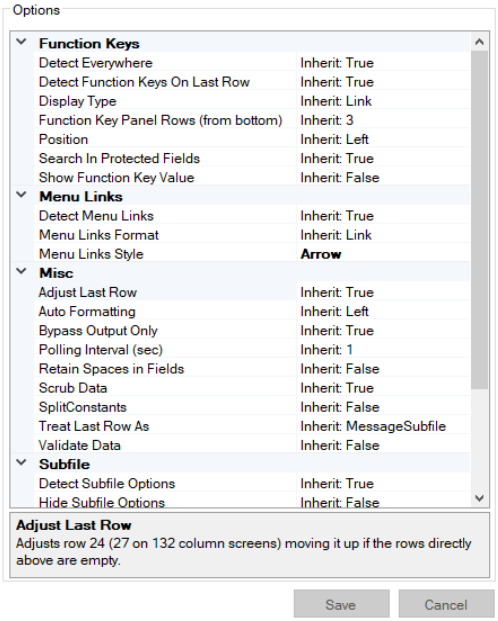


Figure 2-4. Page Options in the Global Rules for Page Options tab.

The integer properties **Function Key Panel Rows** and **Polling Interval** have a drop down which will appear when you click in the field. The only option in the drop down is Inherit, which will apply the default. Clear the field to enter a new number.

It is noteworthy to point out that the font-weight applied to the property values indicates whether it is different from the environment default. If the property's current value is **bolded**, then it is different from the default. If it is a regular font-weight, then the property's value matches the environment default. Inherit is shown in regular font as well since it implies the default will be used.

Save

When you Save or Cancel, all changes will be either saved or discarded.

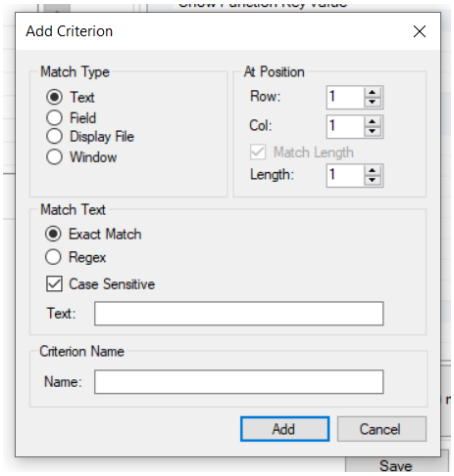
The definition of a Global Rule is stored in the globalrules.json file along with all other Global Rules. Each rule has a path reference to an XML file containing the page options to be applied when that rule matches a screen.

Clicking Save will save all changes to the JSON file, as well as any modified XML files.

For any Rules that have been deleted, their XML files will be deleted on Save. Likewise, Rules that have been renamed will have their old XML files deleted and a new XML file generated matching the new rule name.

Anatomy of a Matching Criterion

Clicking Edit or Add opens the same dialog box to edit a Matching Criterion.



Match Type

The Match Type group box contains the list of possible Matching Criterion types.

- **Text.** Text refers to a text matching criterion that allows you to set the position on the screen (row, column, and length of text) to grab text from. You can choose whether to match that text exactly to a provided value, or use a regular expression to search the string. Matching can be case sensitive or insensitive.
- **Field.** A Field Matching Criterion allows matching input fields at a particular row and column on the screen. You can choose whether or not to also match the field based on length.

- **Display File.** Display File Matching Criteria allow you to match based on the Qualified Display File Name (eg FMV8GEN/FMWVD1KD). Most interactive programs have a DSPF associated and the Qualified Name is available in the Hub while processing the screen. You can match exactly to limit the rule to programs using a particular DSPF file, or use Regex to match on the Library name to match all programs within a particular library (eg XL_PRDEMO).
- **Window.** The Window Matching Criterion is the simplest. It checks whether the screen is a window.

Keep in mind that for Text and Field Matching Criteria, the Row and Column values are tested as Relative positions. This means that the values will be counted from the top left corner of the window, rather than the absolute top left corner of the screen.

Criterion Name

The Name field is required. It needs to be unique within the current Rule, but not across rules. It also needs to have no spaces. These requirements are needed in order to facilitate referencing the Criteria in Custom Filter Logic.

How to create A Global Rule

The following steps outline the intended flow of creating a new Global Rule, however, steps 2 through 7 can be done in any order you feel comfortable with.

1. To create a new rule, click the **New** button in the bottom left. A blank rule will be created and the cursor will be placed in the Name field.
2. Enter a **Name** for your global rule. You will not be able to save until you provide a valid name.
3. Provide a more detailed description of the rule in the **Description** field.
4. Check the **Do not process**… checkbox if you want no further Global Rules to be run against the screen when it matches this rule.
5. Click the Add button to create a new Matching Criterion.
6. Select the AND/OR filter logic for your list of Matching Criterion.
7. Select and define the desired Page Options.
8. You are done. Click the **Save** button in the bottom right corner of the form.

Changes to Global Rules for Page Options will only be shown after the next refresh of any active session. This includes when you modify global rules in the Presto Designer.



Having large numbers of rules/criteria matching against a screen will affect performance, so take advantage of the short circuiting options provided (AND vs OR operations and the "Do not process..." checkbox in order to minimize the number that need to run against any given screen.

Migrating Global Rules for Page Options

If you have separate development and production environments, you may encounter the need to migrate your Global Rules for Page Options between environments. All Global Rules are stored on the IFS under Presto's installation folder, in the "paneloptions/globalrules" subfolder. This folder will contain a "globalrules.json" file, along with all other Global Rules. Each rule has a an XML file containing the page options to be applied when that rule matches a screen.

To migrate your Global Rules for Page Options to a new system, you only need to copy "globalrules.json" and all page options XML files from the development system to the production system.

CHAPTER 9

Chapter 9 - The Presto Visual Editor

After installing Presto we recommend that you take a quick tour of the IBM i applications you intend to make available to your users via a browser. As you navigate your screens you'll quickly get a sense of the types of changes you'd like to make to enhance your applications to be run in a web browser.

For example, you may have data entry panels that - within the constraints imposed by fitting 75 fields from one of your ugliest files into the 24x80 boundaries - are cluttered and difficult to read. Or you may have applications that you've purchased from vendors where the screen layouts don't match your in-house apps. Or you may simply want to do a minor face lift to an old application that is used all the time without cracking open the RPG and DDS. Or you may be wanting to deliver a full blown web application with all the latest bells & whistles, but built on the bones of your rock-solid, tried & tested green screen applications.

Regardless of your ultimate destination, the easiest way to improve your screens is with Presto's visual editor. This chapter discusses this tool and describes the situations in which it is most useful.

Using the Visual Editor

Presto's visual editor allows you to add and work with elements on your page.

To work with the visual editor, click the Visual Editor tab at the top left, as shown here:

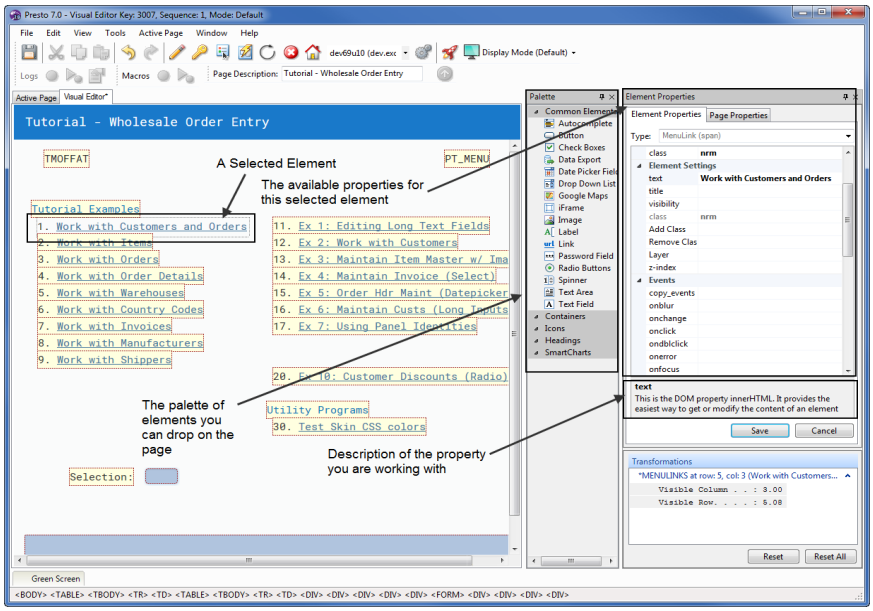


Figure 2-1. Activating the Visual Editor.

In Figure 2-1 after selecting the Visual Editor, Presto loads design tools on the right of the page. On the right are the Properties Grid and Transformation List. Use these controls to apply and work with element properties.



A 'transformation' describes the action that changes a specific attribute of the page element you have selected. So setting the font color of an element to red is a 'transformation' that had been applied to the selected element.

When you work with the visual editor the panel controls are disabled. You can't navigate around or use your function keys, etc.

Saving Visual Editor Changes

To save your changes in the Visual Editor, click the 'Save' button, directly above the summary list of transformations.

Selecting Page Elements

Select an element on your page by clicking it with your mouse pointer. When you select an element the following things happen:

- Presto displays a dotted border around the element.
- The Element Properties area shows the type and attributes.
- The Transformations areas shows any saved transformations.



Note that you can 'tear away' the Properties window from the main Presto Designer window. This is particularly useful if you have multiple monitors and are frequently editing complex screens.

The following sections describe the tools for working with page elements.

Moving Elements

There are a few ways to move elements you have selected in the visual editor:

- **Drag & Drop.** The quickest & easiest way to move elements is by dragging & dropping them with your mouse. To do this, click the element and hold the left mouse button down as you drag it to a new location. When you move things this way you'll notice the snap-to-grid feature which make it easier to align objects on your screen. Hold down the Ctrl key while dragging an element to temporarily disable the guidelines and auto-align (snap-to-grid feature).
- **Drag & Drop Blocks of Elements.** You can also use your arrow or cursor to draw a box around a set of elements on the screen, and drag them to a new location as a group.
- **Arrow Keys.** You can also move one or more elements by clicking them and using the arrow keys to move it a pixel at a time. Hold down the Shift key to move elements 10 pixels at a time.
- **Aligning Elements.** You can also select multiple elements and, while still holding down the Ctrl key, press one of the arrow keys to align all the selected elements to the same baseline. This is a very useful feature, described in more detail below on page 161.
- **Editing the Position Properties.** In the visual editor you can also manually enter a specific position. This technique is useful when you want to align several items to the same column or row.

Note that you can also use Ctrl+Click to select multiple items before moving or changing them!

Moving an Element: Drag & Drop Example

In this illustration we have picked up the 'Utility Programs' text string and are moving it over to the left of where it originally was:

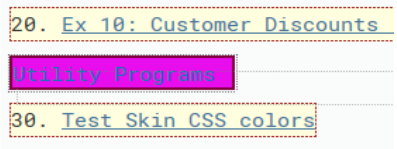


Figure 2-2. Dragging & dropping an element to a new location.

To move elements you just need to drag & drop them. Left-click anywhere within the area contained by the dotted outline then, when you release the mouse button, the element drops into its new spot.

In the illustration above we are dragging the text string and have not yet released it on the panel.

After you drop the element you'll see that Presto has made the following changes:

- **Properties.** The element properties are updated with the results of your change. In this case, Presto updates the Position.



An element's position is represented by the x and y coordinates of its upper left corner relative to the page, with the upper left corner of the page being the origin (0,0).

Almost all the properties correspond directly to CSS (Cascading Style Sheet) attributes. For information on any of the properties, you can click on an entry in the window and press F1.

- **Transformation Record.** The bottom right of your screen displays a list of the transformations you have applied to your panel.

Click the Save button to save your changes, or Cancel to return all the changed elements to their last saved state.

Aligning Multiple Elements to a Common Baseline

The Presto Visual Editor has a useful shortcut for aligning multiple elements to the same baseline (in any direction). This feature is helpful in many design situations with fields, buttons, inputs, headings or other elements.

To use this feature use Ctrl+Click to select multiple elements, then press the arrow key corresponding to the direction in which you want to align them.

Consider this page with several fields:

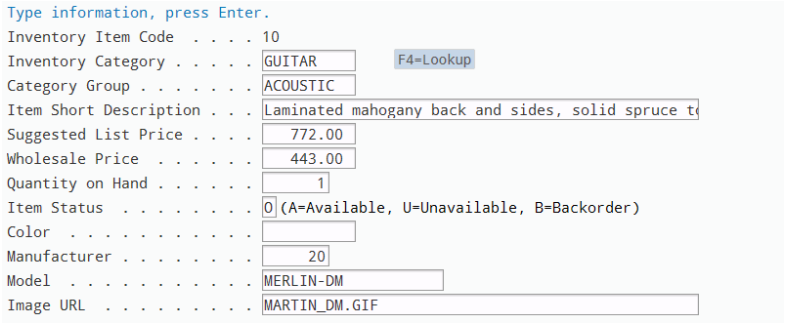


Figure 2-3. Initial screen layout, all fields aligned left.

To align all the input fields to the right, we can just select them all (clicking on each field while holding down the Control key). Pressing the right arrow key (while continuing to hold down the Control key) produces this outcome:

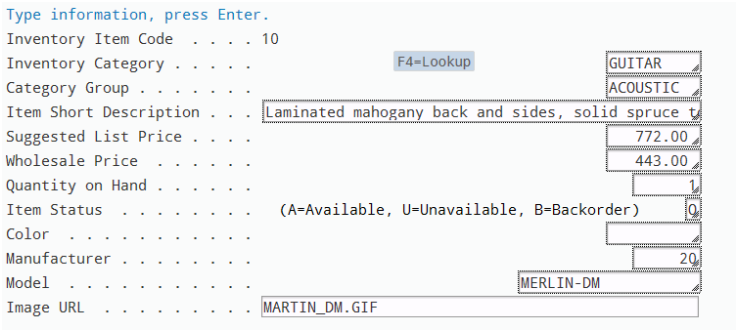


Figure 2-4. All fields now aligned to the right.

While this example is somewhat artificial, the feature is highly useful in situations where you need to align headings in a list, text labels in a page, buttons on a form or any other screen element.

Working with Element Properties

Use the Presto Visual Editor to review and work with element properties. Here is an illustration of this area after clicking a static text string on the panel:

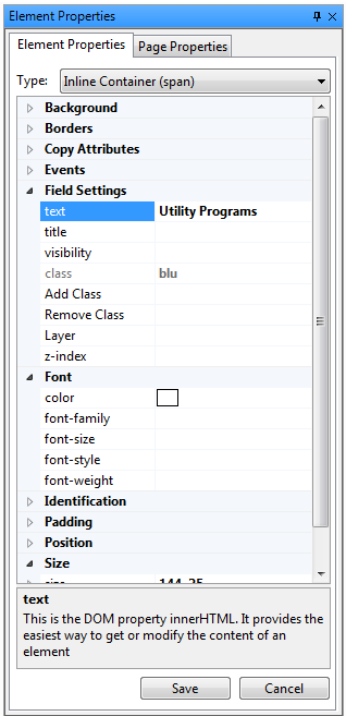


Figure 2-5. Options for working with elements and their properties.

In the illustration above you can see that the element selected is an Inline Container (span). The string's Field Settings and Font attributes are open. The remaining options are set to their default values.

This chapter includes many examples of how to use the Properties menu.

Reviewing, Resetting, Importing and Exporting Properties

Below the window illustrated above is another that lists the property transformations you have added:

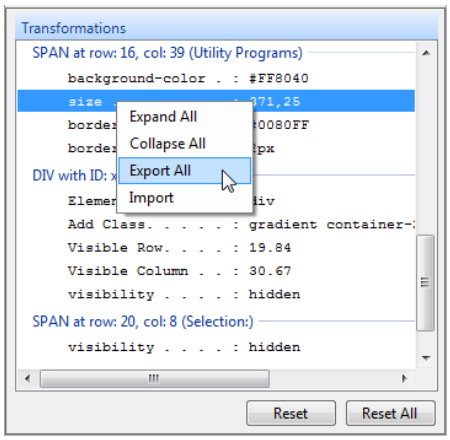


Figure 2-6. Working with transformation entries.

From this window you can review or remove your transformations, as well as import/export the set from/to another page.

Expanding & Collapsing the List

As illustrated above, you can expand & collapse the list of transformations using the right-click options to Expand All or Collapse All.

Resetting Transformations

To reset or clear a transformation select the main entry (the line with the up or down arrow) and click the Reset button.

Resetting Individual Properties

To reset or remove an individual transformation property click the entry (or entries) in the list above and push the Reset button.

Exporting and Importing Transformations

If you have two screens that are very similar, but not identical, you may want to copy a set of transformation from one to the other. You can do this using the Export Transformations option - when you do this, Presto prompts you to save a text file, which you can then import to another page as a starting point.

Note that this is different from the feature available at the Manage Pages interface, where you can export/import all the custom features attached to a page - this option includes only the transformations.

Reattaching Transformations

Transformations are normally associated with fields and texts based on their location in the original green-screen program. If you change this original program after adding custom transformations, it's possible some of your transformations may lose their links to the screen elements. These transformations are highlighted in red in the transformation list:

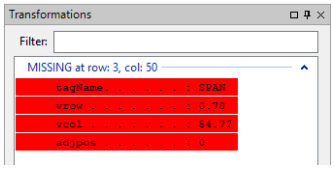


Figure 2-7. Missing element in transformation list.

To reassociate the field with the transformation, click on the transformation in the transformation list and drag and drop it on the field. This updates the row/column on the transformation to the location of the field.

Setting the Element Type

When you select a screen element the Properties area displays the element type at the top. For example, in the illustration below we have clicked on the first menu option:

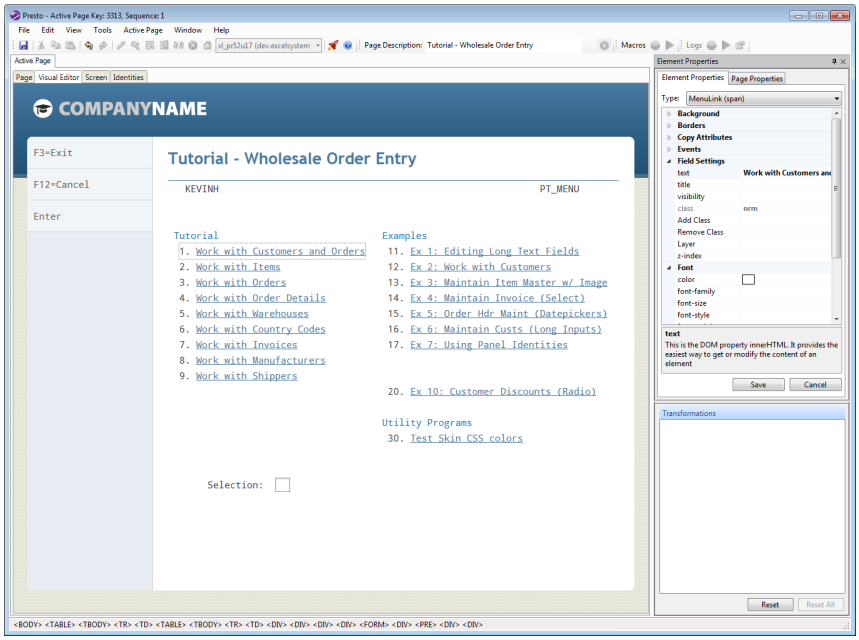


Figure 2-8. Checking the element type.

The screen title is a plain text string and in the web page Presto displays it in an HTML . This is the element type shown at the top right. The text input field at the bottom of the screen is an element of type <input>. When you first install Presto, and before your start applying transformations, all your elements will be of these two types.

You can use transformations to both alter the appearance of an element and/or change an element to a completely different type. Change the element type in the Properties menu or by right-clicking the element and selecting a different type from the Transform To menu, as shown here:

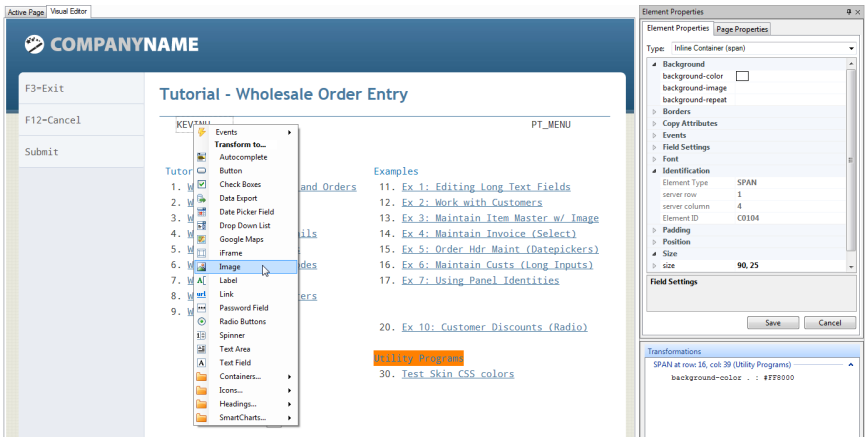


Figure 2-9. Switching an element to a different type.

In this illustration we have right-clicked the user name in the top left of the screen and are in the process of converting it to an image.

The online tutorial for this guide includes two detailed examples:

- "Changing Elements to New Types" at `http://www.myfreschesolutions.com/nexus/apps/PRTUTRL.pgm?tab=chgElmTyp`.
- "Selecting and Changing Multiple Objects" at `http://www.myfreschesolutions.com/nexus/apps/PRTUTRL.pgm?tab=sltChgMltObj`.

Check these examples for further information on this feature.

Transformation List Filtering

The Transformation List in the Visual Editor has the ability to filter. This makes it easier to find a transformation applied to the page:

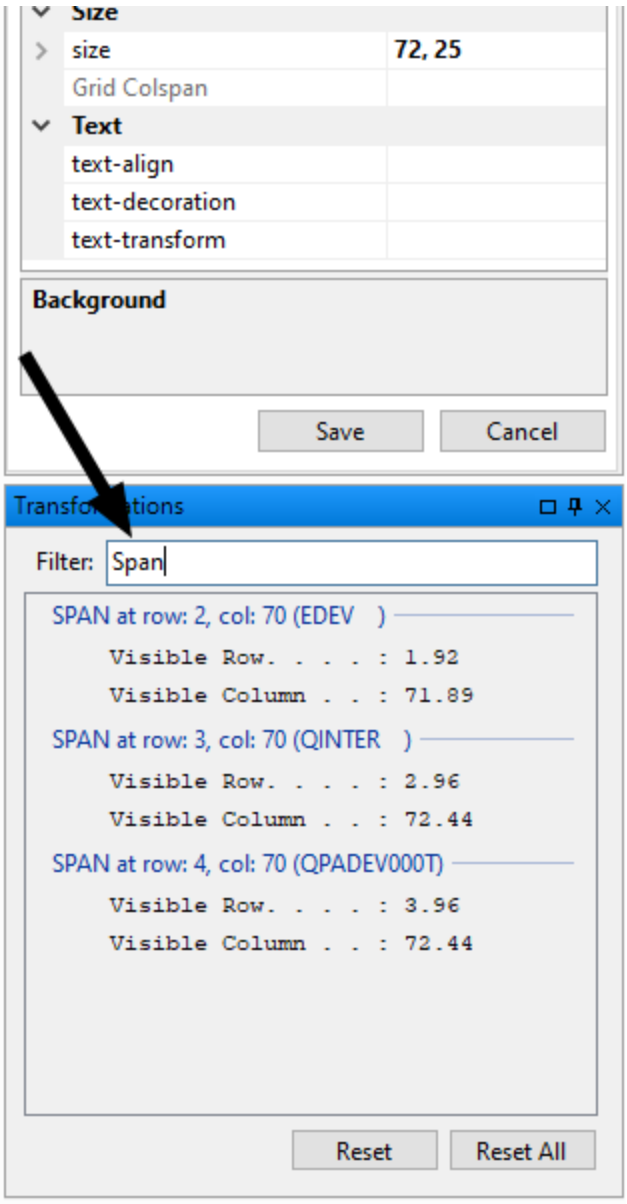


Figure 2-10. Filtering on the transformations list.

Resizing Elements

Presto assigns default dimensions to an element based on the amount of space it occupies on the original green screen display. You can use the Visual Editor to resize screen elements.

To resize a screen element start by selecting it. Then grab the tab in the lower right hand corner and drag it out to the dimensions that you need.



Keep in mind that when you move or resize elements you may cover or partially obscure other page elements! A symptom of having done this is that the new element you have added doesn't work as you would expect. For example, a drop down list won't drop down when you click on it, or a field won't become editable.

To address this you can use the z-index property to set the overlay priorities for the elements on your page.

Example: Shortening Text Fields Using the Element Properties

When Presto creates the text fields on your screen it calculates their length based on the field and page sizes. You may want to adjust a field's size for cosmetic reasons or to free up space on your screen. You can use the Visual Editor options to do this. Consider this page, with several text fields:

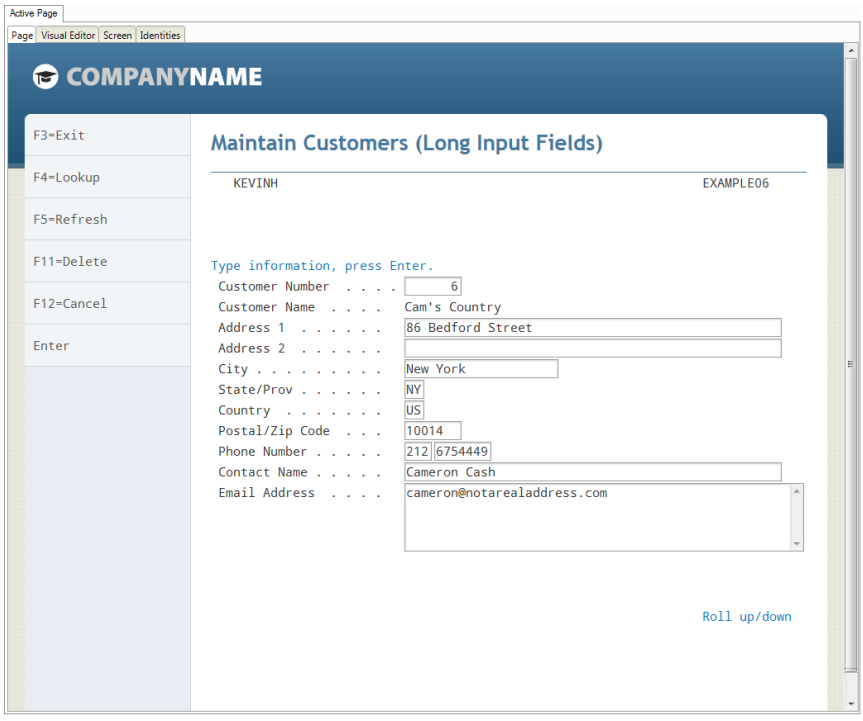


Figure 2-11. A typical flat panel with several text fields.

Suppose you want to add something else to the panel to the right of the Address, Contact Name and Email fields. To make these shorter you can use the tools in the Visual Editor by following these steps:

1. In the Presto Designer, navigate to this page.
2. Click the Visual Editor tab.
3. Click the text field to change (the 'element'). You can hold down the Ctrl key while selecting multiple elements if you want to change them all in the same way.
4. In the Properties window on the right, find the field size attributes.
5. Change the Width attribute for each field.

Finally, save your changes.

In this illustration we have completed the steps outlined above:

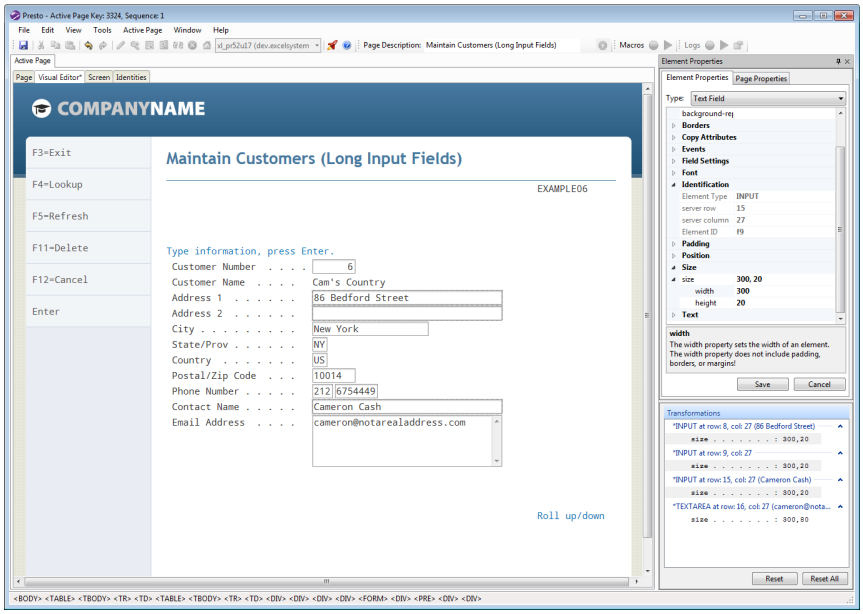


Figure 2-12. Changing field lengths using the Visual Editor.

The Transformations window (lower right) lists the changes that you make, and you can reset/clear them here too.

Example: Using the Element Handle and Edges

You can also resize elements by dragging their edges or the handle in the bottom right corner.

Here are ways in which you can resize an element using the [mostly invisible] handles around the edges:

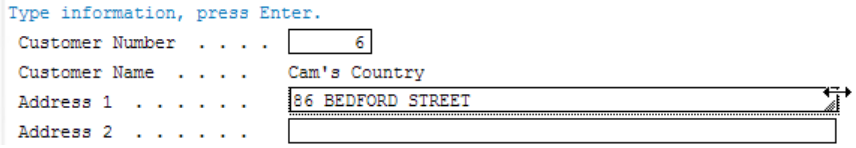


Figure 2-13. Resizing an element by dragging the edge.

Visual editor elements can be resized using the bottom and right edges, and the tab in the bottom right corner:

- To make an item wider or narrow, hold & drag the right edge (as we have done in the illustration above).
- To make an item taller or shorter, hold and drag the bottom.

And if you drag the item by the bottom right corner, you can resize it in any direction.



Note that dragging the top or left edges of an element is tantamount to moving it, rather than resizing, so Presto doesn't enable the resizing on those edges of the element.

Inserting New Elements

You can use the visual editor to insert new elements into your pages. These can range from simple text strings to iframes that you can use to pull in almost any content from your web server.

To insert a new element start by drag & drop an element from the list:

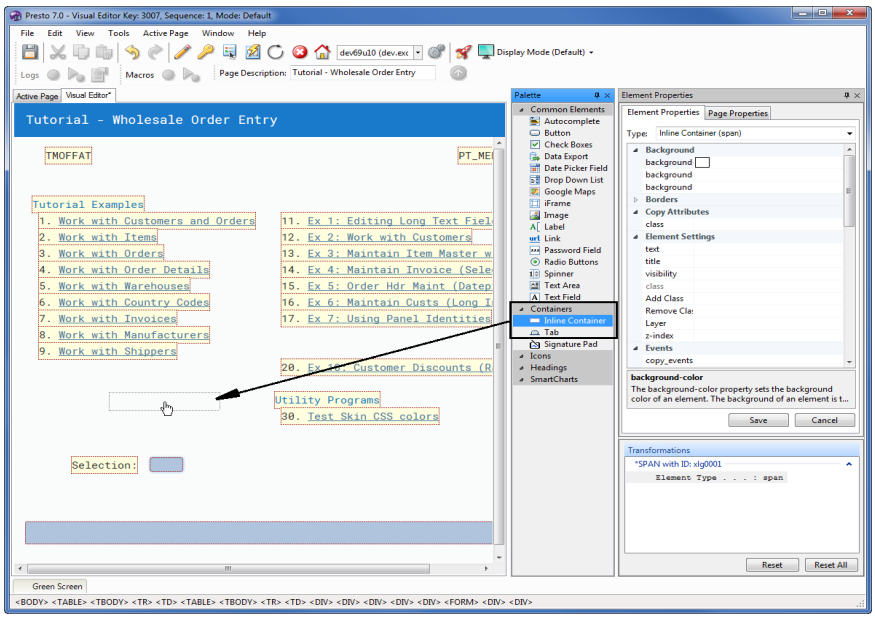


Figure 2-14. Using the palette to add an element to the page.

In this illustration we have right-clicked near the top of the page. When the options appear (with the text Insert... in bold at the top) move your cursor to select an element type.

When you select the element, Presto inserts a placeholder which you can then customize (depending on the type you select). You can use the Properties window to switch this element's type and/or adjust its attributes.

Transformation Entries for New Elements

The other visible change you'll see in the Presto Designer when you add a new element is the transformation entry:

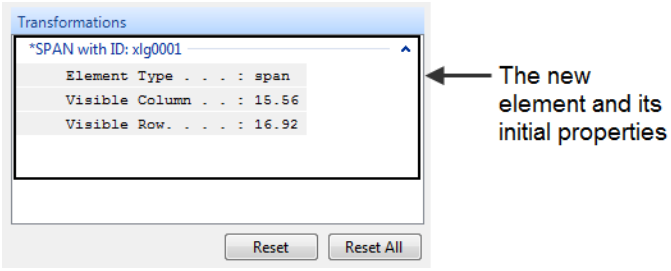


Figure 2-15. The new entry in the Transformations list, showing the added element.

This will also list any changes you subsequently apply to this element.

Removing an Inserted Element

To remove an element click the transformation entry, then the Reset button:

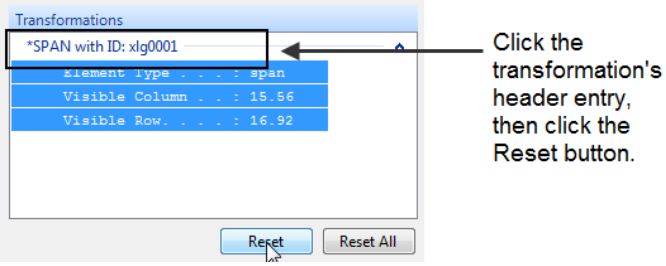


Figure 2-16. Preparing to remove (reset) an inserted element.

The next chapter lists the element types you can insert or transform other elements to.

If you are customizing a page using both the visual editor and custom HTML, elements you manually add to your custom HTML can be subsequently manipulated with the visual editor as long as those elements have appropriate row and column tabs.

For example, if in a custom HTML you manually added this new text string:

```
<span id="C1606" row="16" col="6" >12. Other Programs to Call</span>
```

You would be able to further manipulate it in the visual editor.

Selecting Multiple Items (Group Selection)

Group selection allows you to click multiple items in order to simultaneously apply a common property or set of properties.

You can select multiple items in two ways:

- By holding down the Ctrl (Control) key as you click all the items.
- By drawing a rectangle around the items.

The first method allows you to select multiple items that may be scattered around the screen, interspersed among others. The second method is faster and more fun!

After selecting the elements you can then edit the relevant properties.

Note that Presto will add individual transformations for each selected element.

You can unselect a selected element using Ctrl+click again. Unselect all selected elements by clicking outside of the selected elements while not holding down the Ctrl key.

Technical Details

When you use this feature it is up to you to set shared element properties that make sense. For example, if you select a SmartChart, an image and an input field, Presto will display the properties for the last element selected and not validate whether the option(s) you set make sense for all the selected elements.

Editing Tables

You can use the visual editor to work with the properties of your HTML tables. You can work with individual elements (text strings or other elements) within tables as well as edit the properties for the table as a whole (to move it, resize its outside edges, or adjust the border).

Note that in this release you cannot yet use the visual editor to insert new tables, or select individual table cells or rows.

Using Screen Values in the Visual Editor

While working in the visual editor you can also view a green screen preview, and use screen values as part of your transformations:

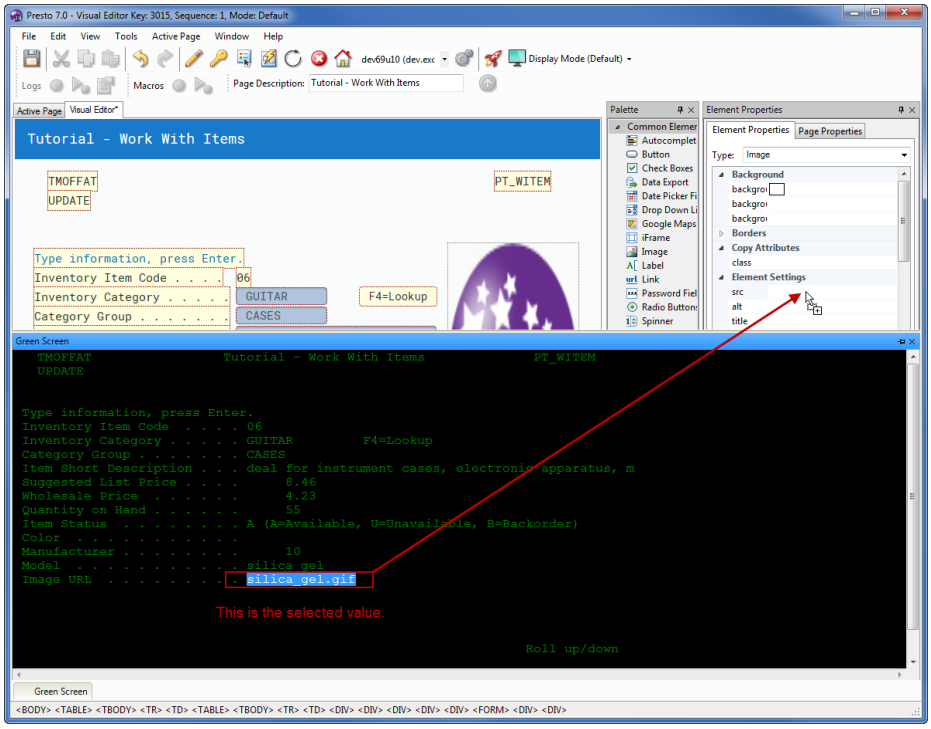


Figure 2-17. Green screen view accompanying the visual editor.



If this isn't visible in your current Presto window you can enable it via the View>Screen menu option. The preview can be attached to the main Presto window or left floating as a separate window, as shown here.

From this area you can drag fragments to your visual editor settings, to set values in this syntax:

```
{screen([row],[column],[length])}
```

For example:

```
{screen(8,29,52)}
```

The next pages have examples of this.

Example 1: Using Screen Values as Autocomplete Parameters

One of the ways in which you can use screen values is to populate autocomplete parameters.

In the following illustration we are setting up an autocomplete that will limit a list of cities (where the user starts to type the city name) to a particular state. The city is the field retrieved, but the state will be the additional parameter:

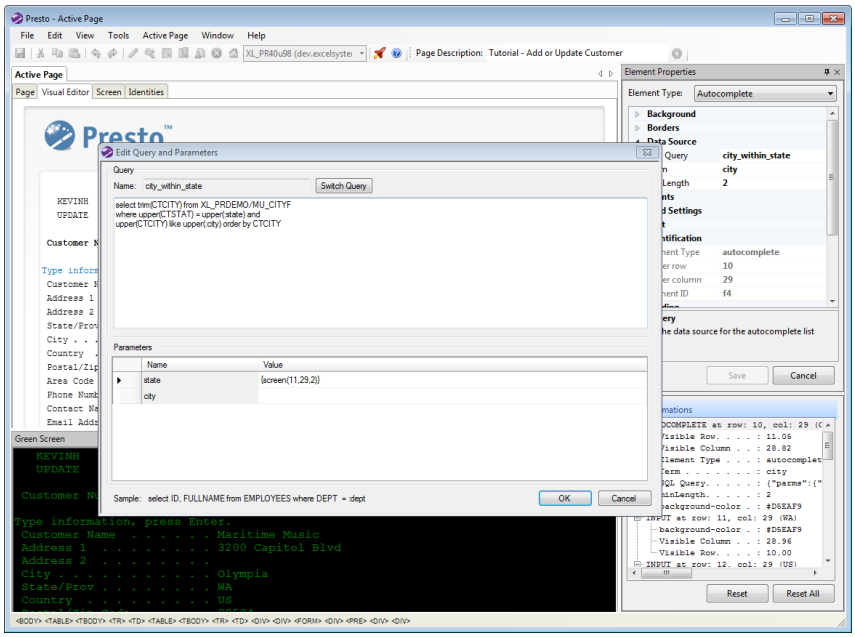


Figure 2-18. Dragging the state field from the screen to the query parameter value.

Because the state field will vary for every customer, the value needs to be retrieved dynamically from the screen whenever the customer's data is edited. In the illustration above we have performed the following steps:

1. Selected the visual editor.
2. Ensured that the field we need to use is visible in the green screen view.
3. Changed the 'City' field on the screen from a text field to an autocomplete, using the following SQL query:

```
select CTCITY from XL_PRDEMO/MU_CITYF where upper(CTSTAT) =
upper(:state) and upper(CTCITY) like upper(:city)
```

   The value 'state' is the extra parameter to be retrieved from the screen.

4. Then we clicked in the green screen preview and highlighted the state field. Once it was highlighted, we dragged it up to the Parameter Value field.

When we did this, Presto populated the Parameter Value field with this:

```
{screen(11,29,2)}
```

As a final step you can save the parameter, and click OK to close the Query Selection window.



Note that you can also type the screen value parameter yourself; you don't have to drag & drop it from the green screen view (though that is usually easiest).

Example 2: Using Screen Values as Element Properties

You can also use screen values as element properties. For example, suppose that you want to use a screen value to set a CSS class for several screen elements. In this example we are customizing

the Work with Items screen from the tutorial application. Our items belong to different inventory categories, and we want to set some field attributes depending on the category.

To do that we have selected all the items that will have a custom appearance based on the inventory category, and are dragging & dropping that field value into the visual editor's 'Add Class' property:

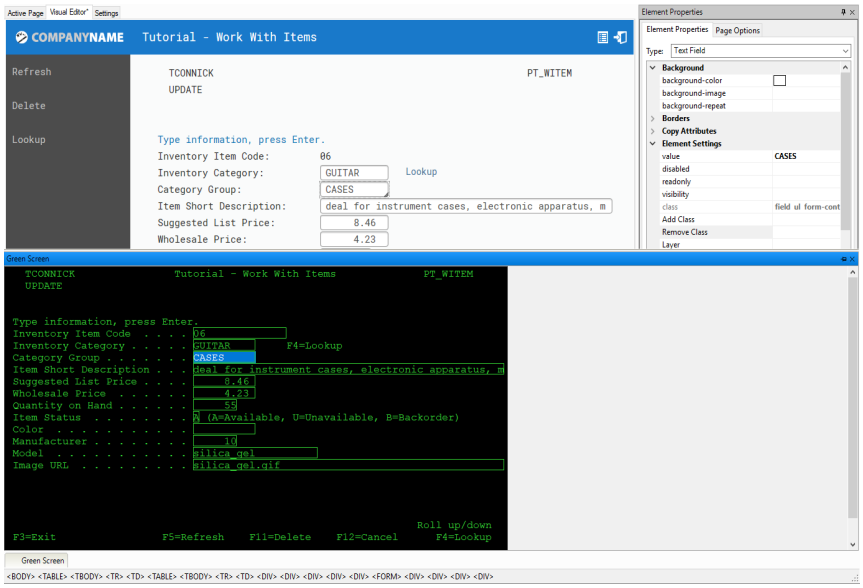


Figure 2-19. Using a screen value in your element properties.

In this illustration you can see that we have selected multiple elements on the screen (by enabling this option using the feature described next). Then we highlighted the Inventory Category field on the screen, and dragged it up to the Add Class property.

This is the outcome in the designer:

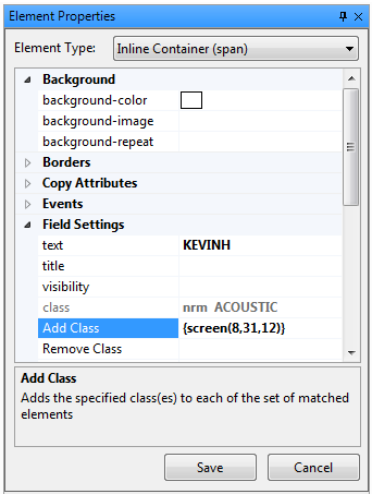


Figure 2-20. Adding a screen value as a CSS class.

In this illustration you can see that the Add Class property includes a screen variable, and in the class property the value ACOUSTIC tells us that that's the value added in this particular instance.

The following transformations are recorded:

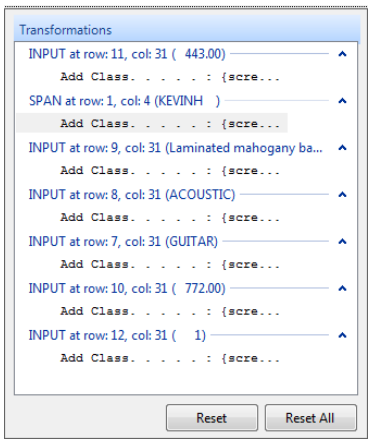


Figure 2-21. Setting dynamic element properties, based on screen values.

What this means is that the CSS class for these items will be dynamically set at run time, depending on the Inventory Category of the item that the user is working with.

This is a simple example of this feature, but the possibilities are huge!

Visual Editor Insert Elements Palette

While working in the Visual Editor you can also use the palette to add screen elements. Here is an illustration of Presto, showing the palette at its default starting location:

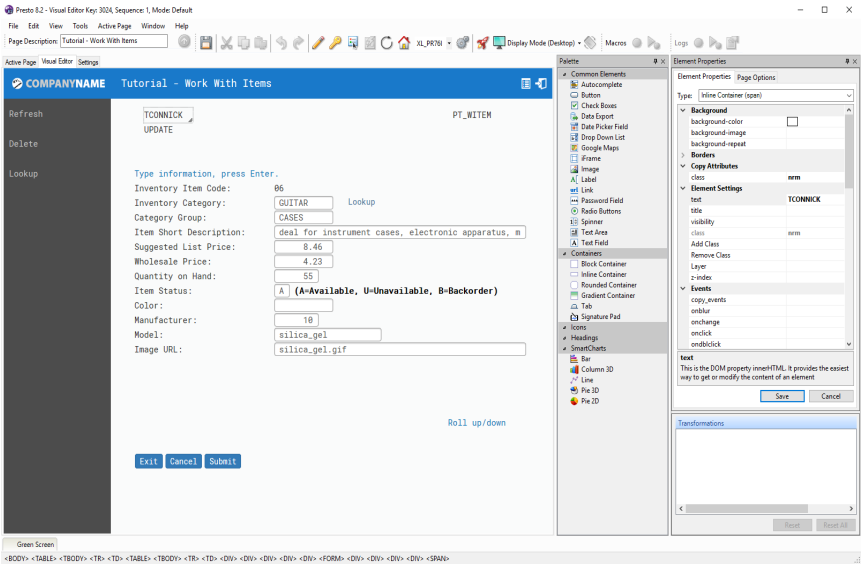


Figure 2-22. Visual Editor Palette feature.

You can drag & drop elements from the illustrated list onto the screen, then further adjust the initial, default properties. The palette includes a wide range of elements as well as variations on the most useful types. Using the palette reduces the number of steps it takes to define custom screen elements.



The palette is initially shown to the right of the main design window but you can move it around the Designer to other locations as well, or tear it away from the main window entirely and have it outside the main design tool.

See Figure 2-23 below for an illustration.

Adding Custom Palette Entries

The default library of palette entries is the file palette.xml, on your PC. You will typically find this file at ESDI\Presto\Visual Editor\, or Program Files (x86)\ESDI\Presto\Visual Editor.



Presto does not retain changes to this file on upgrade. If you install the next version to a different directory the file isn't touched, but if you want to add custom palette entries, don't change the default set.

To add your own custom palette entries and retain them as you upgrade, create a new file named palette.xml on your PC at the following location:

```
C:\Users\[User]\AppData\Roaming\ESDI\Presto\palette.xml
```

You can create the new file with Notepad or another plain text or XML editor.

Here is a simple example of the XML to create a new custom category and element:

```
<palette>
        <categories>
                <category id="customKJH"/>
        </categories>

        <category id="customKJH" title="CustomKJH" image_id="category">
                <template id="button-1"/>
        </category>

        <template id="button-1" element_type="button" title="Button" image_id="button">
                <attr name="value" value="Button #1"/>
        </template>

        <mode id="responsive">
                <template id="button-1"/>
        </mode>
</palette>
```

The easiest approach is to copy samples from the default palette elements. For example, if you often add containers around groups of elements, the palette makes it easy to create and use multiple container styles of various colors and border styles, with rounded corners, gradients and other characteristics in common, without having to remember particular class names every time you use them. The palette also makes it easy for you to have a library of button and icon styles to add to your pages, not to mention other types of images, and variations on all the other elements you now use!

If there is a particular style of element that you are adding frequently to your pages, and you're not sure how to add it to your palette, please call Technical Support! This is another 'growth'

area of the software and we welcome suggestions on how we can continue to improve it to make it more useful.

Reattach Transformations to Moved Elements

If you are modernizing an application that is still under some degree of active development on the green screen side you may find that a panel change has ‘orphaned’ one or more of your transformations. When this happens, the Presto Designer will highlight the orphaned entry, and you can drag & drop it to the element’s new location.

Consider this illustration:

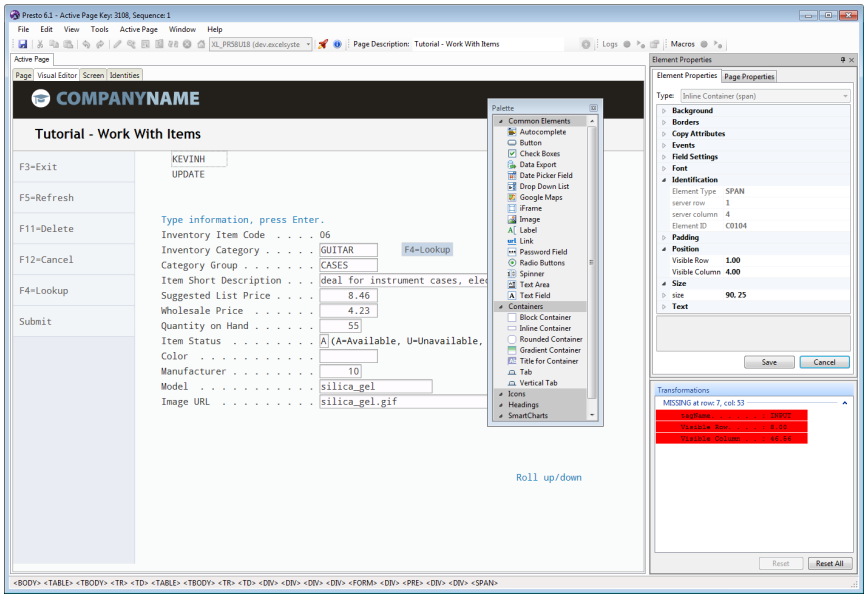


Figure 2-23. Orphaned transformation.

In the lower right you can see that the single transformation attached to this page has been ‘orphaned’ by a change to the underlying screen - the field was moved when the page was updated.

To resolve this you can just drag & drop the transformation from the list on the right to the field’s new location. As you’re dragging, the Presto Designer will highlight your target in green:



Figure 2-24. Reattaching a transformation to a field.

Release the transformation over the target field.

Copy Transformations between Pages

You can create a transformation as a copy of an existing transformation.

The Presto Designer allows you to copy and paste transformations between pages.

1. Right-click a transformation in the Transformation list and select "Copy Selected.
2. Go to a new screen, and right-click in the Transformation list again and choose "Paste Transformation".

Here is an example:

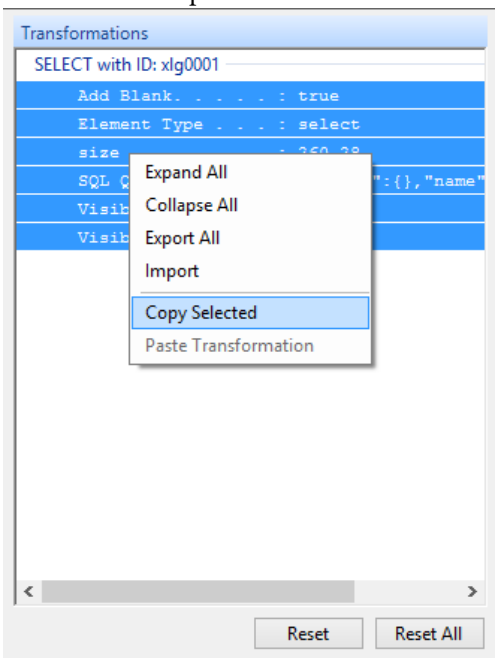


Figure 2-25. Copying a Transformation.

Here is an example of pasting a transformation into the Page's transformations:

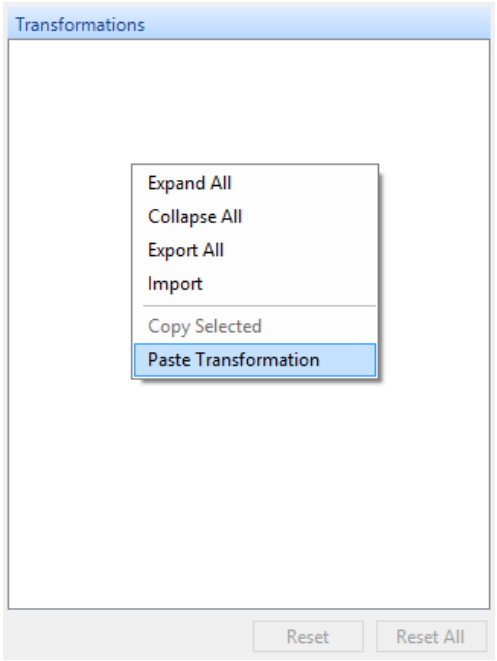


Figure 2-26. Pasting a Transformation.

Here is an example of a transformation pasted into a Page's JavaScript:

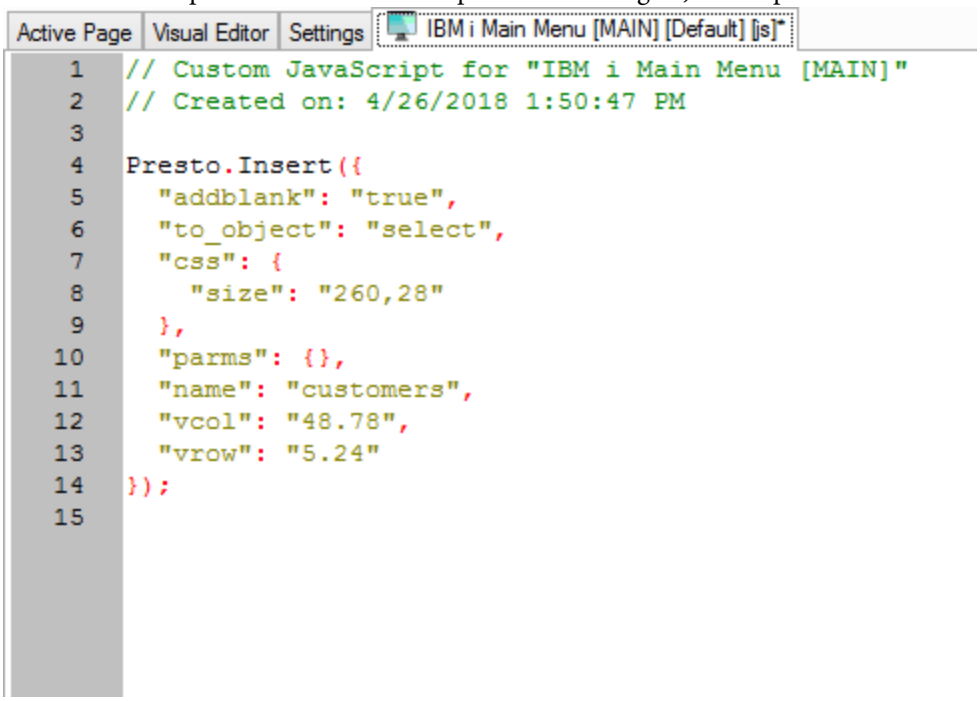


Figure 2-27. Pasting a Transformation into JavaScript.

Notes:

- The copy has no link to the original; changes to the original has no effect on the copy and vice versa.
- If you have multiple instances of the Presto IDE open, you can copy and paste between sessions.

Chapter 10 - General Transformation Properties

This chapter describes general properties you can use when working with most of the supported element types. The general properties range from background colors and border styles to visibility and font. The following chapter describes properties that are unique to a few elements.



Almost all of these generic properties directly or indirectly map to CSS attributes. For information on any particular property position your cursor on that line and press F1. Presto includes links to the W3Schools site where you can find authoritative information on each one.

In the sections below we have included some additional considerations when working with these properties.

Saving Element Transformations

After you add a transformation, click the Save button to store your changes, or Cancel to return to the last saved state.

Once you save your changes, these will be available/visible to someone running the program via a browser.

How Does it All Work?

The transformations on your page are all driven with JavaScript. For each element that you transform the IDE creates a text representation that can be passed to the Presto.Transform function and which is then interpreted by the JavaScript engine as a JSON object (JavaScript Object Notation).



See , Presto.Transform Function Reference for a detailed discussion of the syntax and semantics of the selectors and properties available.

Changing Colors

A number of the available element properties are related to colors. To change any of these, select the desired property by clicking the line it appears in:

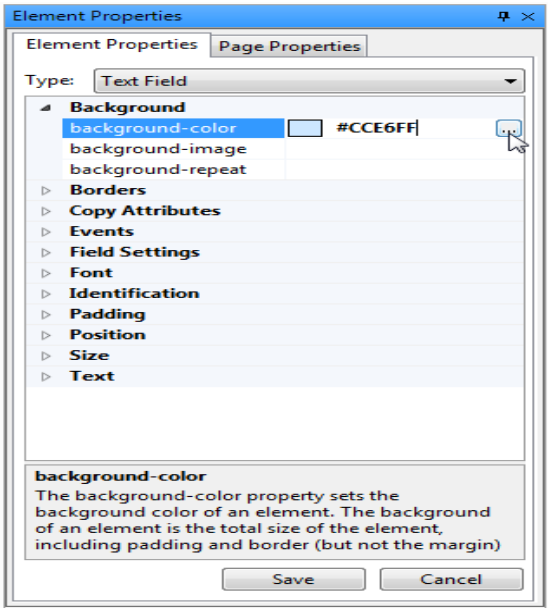


Figure 2-1. Getting ready to change the background-color.

When you click the button indicated above, this color picker is presented:

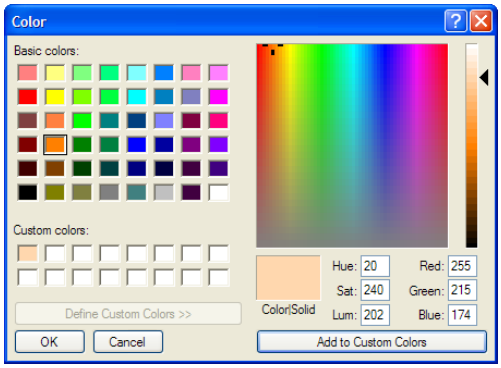


Figure 2-2. Using the color picker to select a color.

Find and select a color that suits your needs, then click OK to confirm your choice.

Selecting a File

Some element attributes, such as background-image, require a file.

To select the file, click the Browse... icon to the right of the attribute name:

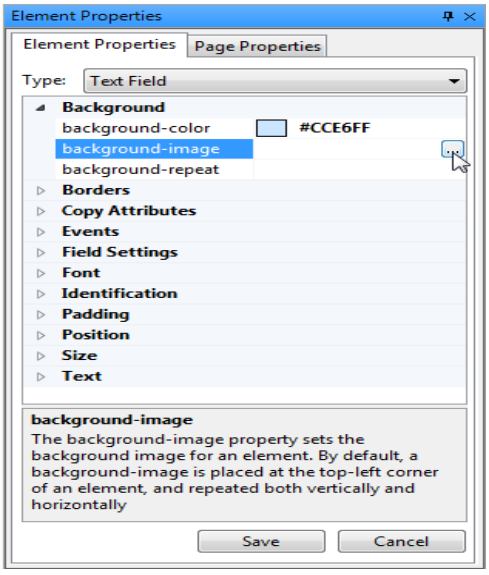


Figure 2-3. Preparing to browse to select a file.

At the 'Open' dialog locate the file to use. In the case of the background image, the file name is added in this format:

```
url('customimages/tablebackground01.jpg')
```

This assumes that the folder \customimages\ exists off the root path for your Presto environment.

Events

When working with elements in the Visual Editor you can assign specific Events[1] to occur when the user performs a specific action. You can use this technique to bind any type of event you want to the HTML elements on your pages. You may want to always perform a particular action when a user checks a checkbox, clicks a link or changes the value in a field.

These events are coded in the Visual Editor using JavaScript code without <script> tags.

The available events are illustrated on the right of this image:

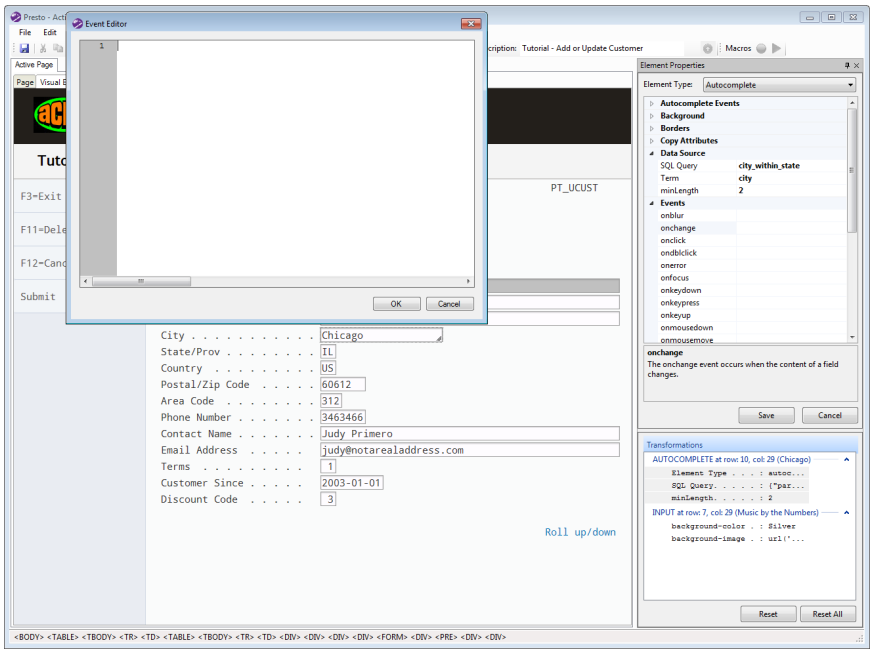


Figure 2-4. Adding an onchange event.

This is a more advanced editing technique but if you have ideas on how to use it and need assistance, our Technical Support team can help you out.

The next sections include two examples.

Example: Triggering an Alert When a User Mouses Over an Element

For example, to trigger an alert when someone mouses over an element start by selecting the element in the Visual Editor. Then, in the Properties tab, click on the onmouseover event and click the browse (...) option that will appear in the area indicated by the second arrow below:

[1] http://www.w3schools.com/js/js_htmldom_events.asp

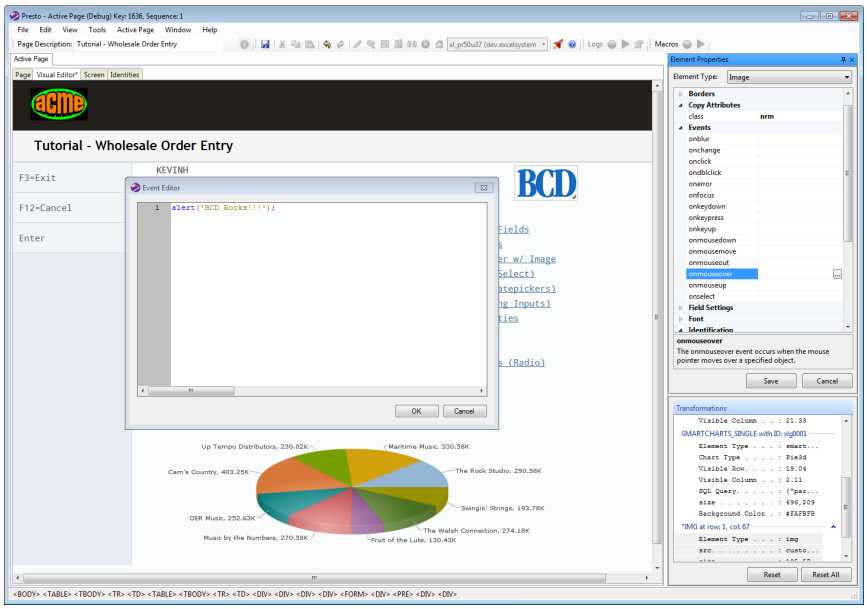


Figure 2-5. Creating an event in the Visual Editor.

In the Event Editor that pops up, type your JavaScript code. In this case, it is:

```
alert('BCD Rocks!!!');
```

Click OK.

Save your changes and navigate back to the Active Page. You will see that when you mouse over that element, an alert pops up with your message in it:

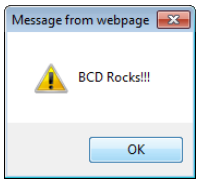


Figure 2-6. Adding an alert when the user mouses over an image.

Feel free to add this feature everywhere!

Field Settings

Several element types support a similar set of field settings. Below are descriptions of these common properties.

The Text or Value Field Setting

You can access the text or value field property for several types of elements. To enter a field value, just type it directly on the line, as shown here:

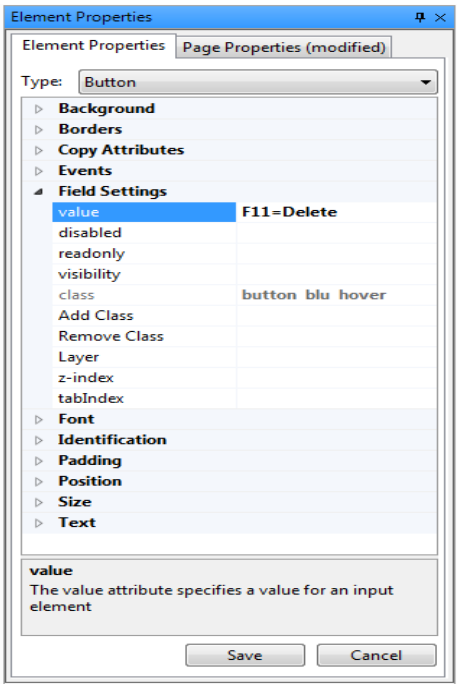


Figure 2-7. Entering a field override value.

You can also do something similar to this when overriding the value of text strings.



Note that Presto has no built-in validation for this text. This means you could select a numeric field in the visual editor and assign it an alpha value. This wouldn't be apparent as a problem until you ran the program.

Setting the Tab Index

You can set the order in which elements are tabbed to by adjusting the Field Settings tabIndex property. The lowest number that you enter is visited first, and so on.



The tab index field setting is not available for divs, spans and SmartCharts.

When you change an element's tabIndex you are overriding automatic settings assigned by your browser as it loads the page and there are some things you need to take into account:

1. You cannot use the tab index value to override which element the cursor is on (or has focus) when you first load the screen.
2. When you adjust the tab index for an element you must also override the tab index value of every element that would by default be tabbed to before the element you override.

The reason for this is that a default tabbing order (top to bottom) applies. If you want to reverse that natural tabbing, or insert a new element in the tabbing order, you need to explicitly specify a tabbing order for all items that appear on the page previous to the new one(s).

To clarify this, consider this page:

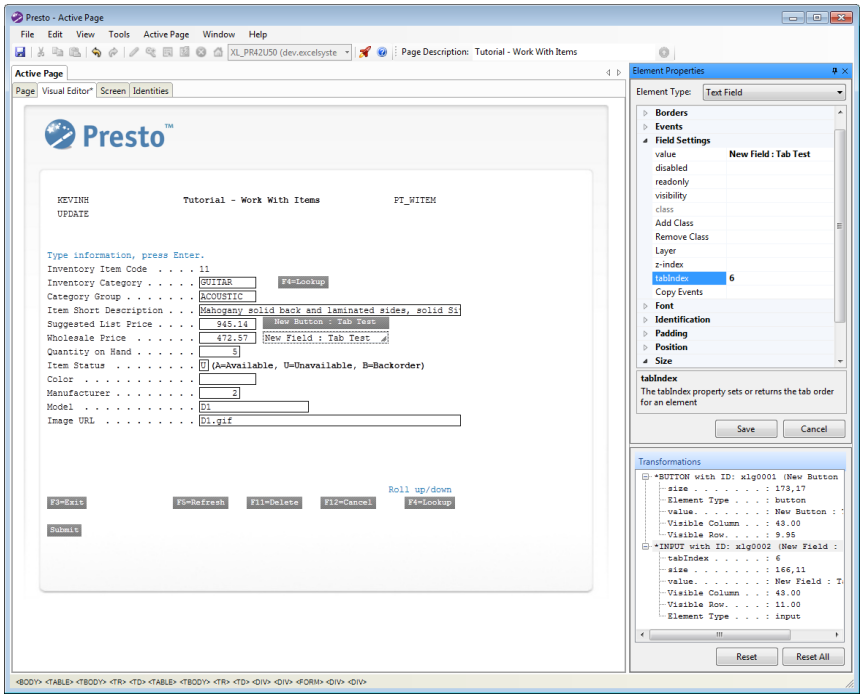


Figure 2-8. Illustrating how to set the tab index value.

Here we have used the visual editor to add the 2 indicated elements to the page. The first step is to assign a tab index value of 1 to the first element on the page, and then a sequential value to every other element that would normally be in the tab order before the elements we've inserted. So to make the panel work naturally, we assign these tab index values (note numbers):

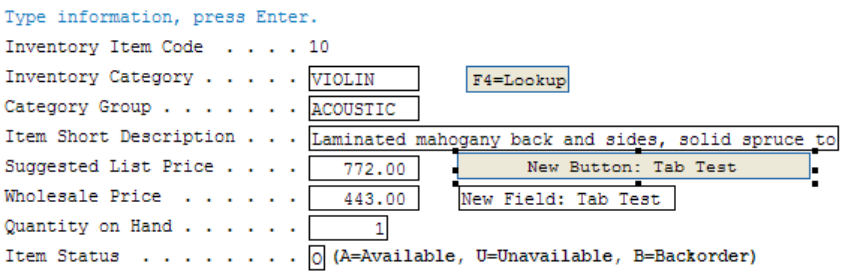


Figure 2-9. Adjusting tab index values to account for two inserted elements.

The second (last) inserted element is the one with text 'New Field: Tab Test'. After this, you can let the page take back over the tab index order.



Depending on how many new elements you have inserted in the page, the quickest solution may be to select all the elements before your first inserted element and assign them all a tabIndex of 1. Then assign your first inserted element a value of 2 (or any value higher than 1). For the elements that share a tabIndex value, the natural order of tabbing will apply.

So, referring to the example above, an alternative (and perhaps more efficient) strategy is to assign a tabIndex of 1 to items 1 through 5, then the new button could have a value of 2, the Wholesale Price field a value of 3, and the second inserted element a value of 4. The remaining elements on the page don't have to be touched.

HTML5 Input Types

Use the Field Settings 'type' property to take advantage of HTML5 input types:

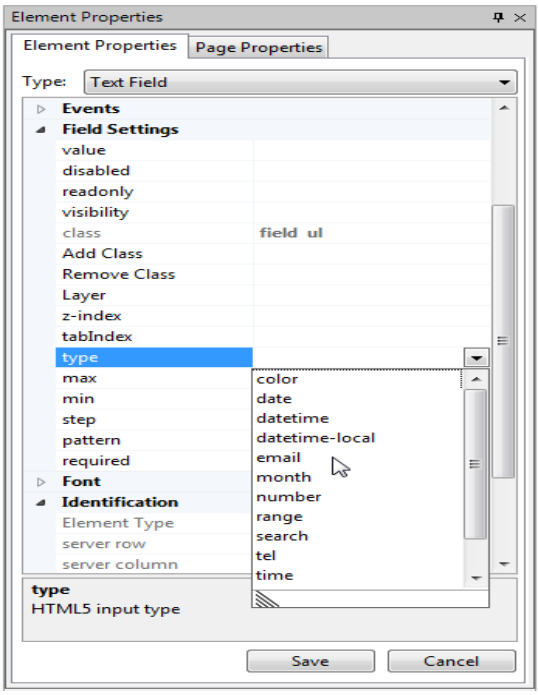


Figure 2-10. Selecting an HTML5 input type.

Search online for additional information on HTML 5 inputs, where you will find many useful examples.

The Class Attribute

The class attribute is noteworthy because it allows you to link elements on your screen to a custom CSS file. This means that when you create a custom skin for your site you can define CSS classes in the stylesheet.css file and then conveniently tag elements via the visual editor to apply those changes.

Here are the steps you need to follow to use this:

1. From the Tools>Modify Skin>stylesheet.css menu option open the style sheet for the skin you are using.
2. Add your custom tags. For example, we added this one to the bottom of our stylesheet.css in our custom skin from the tutorial:

```
/* custom CSS tags Corporate Skin          */
/* FRD 20151222 added .bigblue             */

.bigblue
{
        padding: 0px 1px 0px 1px;
        margin: 1px 0px 1px -2px;
        line-height: 17px;
        border: 7px solid blue;
        font-size: 15px;
}
```

3. Save your changes to stylesheet.css.
4. Open Presto and navigate to the page where you want to apply the attribute.
5. Select the Visual Editor.
6. Select the element.
7. Find the Add Class attribute in the Field Settings.
8. Type the class name (bigblue).
9. Tab away from the field or press Enter.

The change is applied the next time you load your site (the CSS is only loaded when you first connect).



If you are familiar with CSS you will realize that this adds an incredible level of flexibility to your design process. If you aren't yet familiar with CSS, we encourage you to take some time to look into it!

The file stylesheet.css includes many examples you can clone to add your own CSS tags, and a Google search will yield examples of anything else you can imagine you would like to add. And of course, you can call Technical Support and we can give you a hand as well.

Identification Properties

These properties display the original IBM i screen location of the selected element (these are illustrated in Figure 2-11 below). These are useful in custom scripts when you need to identify a particular field on your green screen program, to transform it or retrieve it's current value with Presto.Select.

Padding Properties

These properties allow you to control the amount of white space around an element. These are standard CSS attributes.

Position Properties

The Position properties include the following attributes:

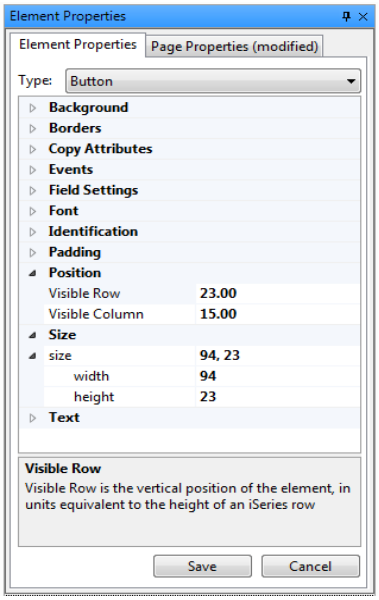


Figure 2-11. Position attributes.

You will most often move elements by dragging & dropping. In these cases Presto updates these attributes automatically, but you can also use these properties to manually fine-tune an object's location and dimensions. To change the values, retype the X, Y, Width or Height settings.

Text Properties

Use these to adjust various attributes for text strings.

Copy Attributes

When you convert a screen element to a different type the existing CSS attributes are by default retained. However, if your new element is of a completely different type (for example, if you've converted some text to radio buttons) it's possible that some of the attributes of the original element are no longer relevant.

To remove all these attributes and start with a clean slate, set the Copy Attributes class value to false:

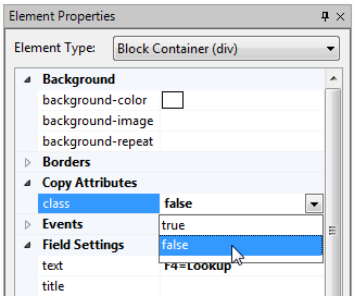


Figure 2-12. Setting the Copy Attributes class to false.

The default value is true.

CHAPTER 11

Chapter 11 - Visual Editor Transformations

This chapter describes each of the element types supported by the visual editor, how to use them, and their distinct properties. Generic properties (such as colors and padding) are discussed in the previous chapter.

Where possible we have included references to more extensive examples in the online tutorial for this guide.

Autocomplete

Autocomplete transformations are particularly useful in places where you have traditionally relied on window lookups to help users populate a field value. Rather than forcing the user to go out to a separate window to select a warehouse code they probably at least partially remember, you can allow them to start typing in one or more characters of the value they're looking for and then select from a subsetted list of entries loaded dynamically to the screen.

This is a powerful feature with some obvious characteristics, but also some more nuanced functionality that you can experiment with to optimize your page's speed when you load subsets of entries from larger files.

The following sections discuss the options available. There are also some features you can add via custom HTML, described in the section "Enhancing Autocompletes Manually" on page 301.

Overriding the Autocomplete's Styling

Presto autocompletes are implemented using the jQuery UI. You can override the default settings with custom CSS, to adjust the look & feel.

Switching a Field to an Autocomplete

To switch a field to an autocomplete start by navigating to the page and switching over to the visual editor. Then right-click the field and select Transform To>Common Elements>Autocomplete:

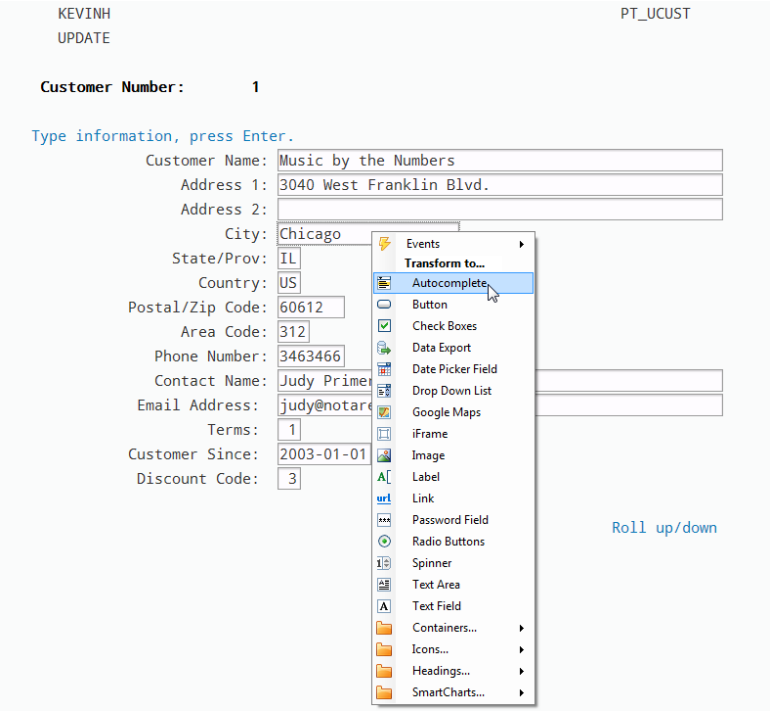


Figure 2-1. Switching an input field to an autocomplete.



You can also switch the element type using the Element Properties menu on the right of the screen.

When you select 'Autocomplete' from the list of element types, the visual editor adds a new Data Source group of properties to the top of the list. These include:

- **SQL Query.** A query you use to generate the list of matching records. Note that if you set up your SQL to retrieve two fields, it will return the first to the program, and use the second for display.
- **Term.** This is the value being typed by the user.
- **minLength.** This is the minimum number of characters the user must enter before triggering the query to load the list. The default value is zero. This option is described below.

The following sections illustrate and describe these options in detail, beginning with the simplest (the minLength).

Setting a Minimum Number of Characters to Enter Before Querying

In some cases, for an autocomplete to be useful you need the user to enter at least a couple of characters to have any likelihood of finding a match. For example, if you have 100,000 names in your database, and the user starts typing 'S...' you don't necessarily want to immediately trigger a query that returns a list of every record starting with S.

To address this you can use Min Length option to set the autocomplete to not query the file until a minimum number of characters have been typed in:

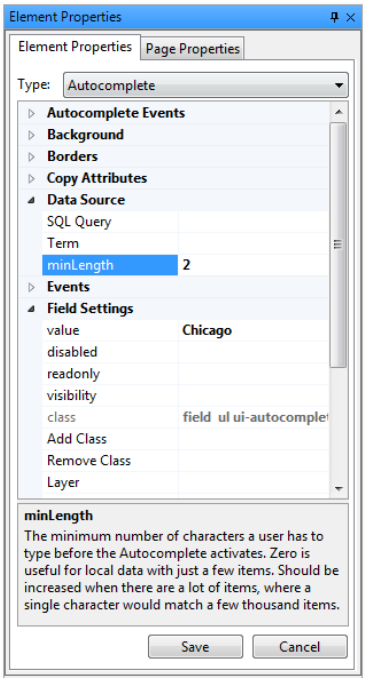


Figure 2-2. Using the minimum length parameter on autocompletes.

You can leave this value blank (or zero) for smaller files or record sets, but if you are looking for matches on large files it's a good practice to set it to at least 2 or 3 characters.

In deciding how many characters to force the user to enter you need to evaluate your user's familiarity with the database and the number of possible matches for the particular field.

Creating and Attaching a Query

The first critical step in enabling an autocomplete is to create and attach a query. In the following sections below we'll discuss the various strategies available when defining the query, but let's begin by just looking at how you would attach it.



The tutorial includes step by step instructions for implementing this example, in the section "Adding Autocomplete Functionality to Input Fields" at `http://www.myfreschesolutions.com/nexus/apps/PRTUTRL.pgm?tab=addAC`

Begin by clicking the button to select your SQL query:

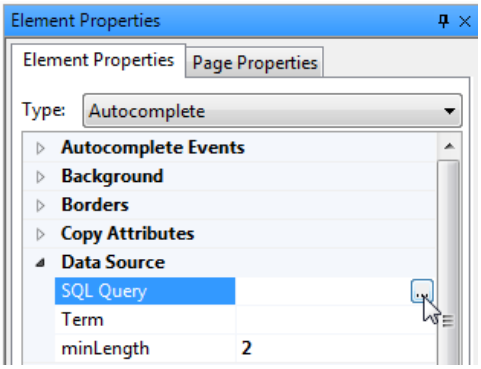


Figure 2-3. Preparing to select the Autocomplete Data Source.

You can also right-click the field on the screen directly:

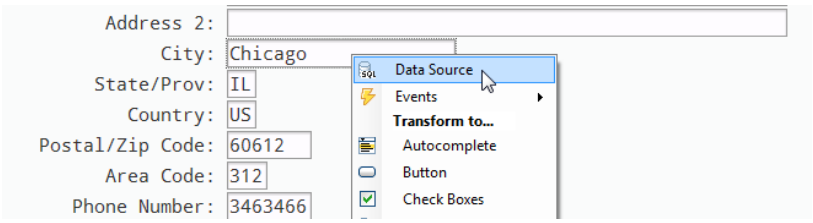


Figure 2-4. Right-clicking an autocomplete element to add a query as the data source.

When you use either of these options Presto presents the Add Query and Parameters window (shown below).

For your autocomplete create a new one in this window or click the Switch Query button to select from the list of existing queries:

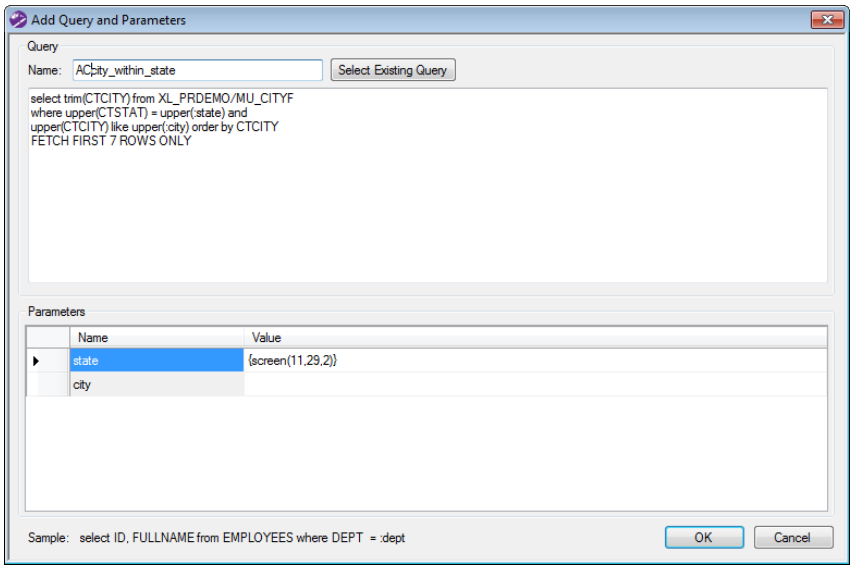


Figure 2-5. Adding a new query.

Enter a unique query description and add your query. Press OK to save it.

Once you've added a query you can close the SQL Queries window.



You can set your SQL to return two values, to search on one, but return the other to the program. The first value is the one returned to the program, and the second is used for the visible search shown to the user. For example, you could allow the user to search for a city using the actual name, but return a city code to the program.

Note that the bottom half of the window has the required parameters. In this case, the State value would be populated based on a value that had already been entered on the screen:

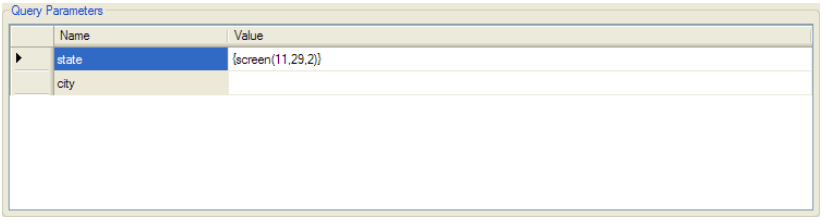


Figure 2-6. Populating the autocomplete's query parameters.

The 'city' parameter is the one that the user would be typing directly, and corresponds to the Term option in the element properties. You can find more information about this in the examples that follow.

Once you select your query Presto lists it in the element properties:

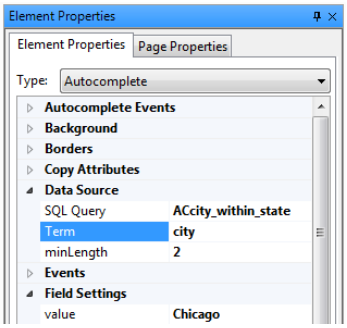


Figure 2-7. Query attached to your autocomplete.

The following sections discuss this feature in more detail.

Selecting Existing Queries

Queries are saved on the server and displayed in this window, which you can access from the Switch Query button in the Add Query window, or from Tools>SQL Queries:

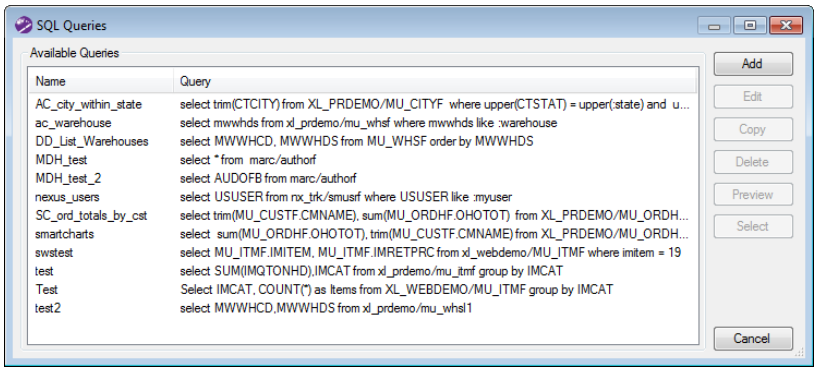


Figure 2-8. The SQL Queries window.

Manage your queries from this window.

Query Examples

Using Unqualified SQL Queries to Load Records Matching in ANY Position

If you use an unqualified SQL query with no 'where' clause, such as this one:

```
select CLCLNA from LG#LIB/LG#CLTL2
```

the autocomplete first loads the entire possible record set, then uses JavaScript to return results within that set, where the matching character values are found anywhere in the field. Consider this illustration:

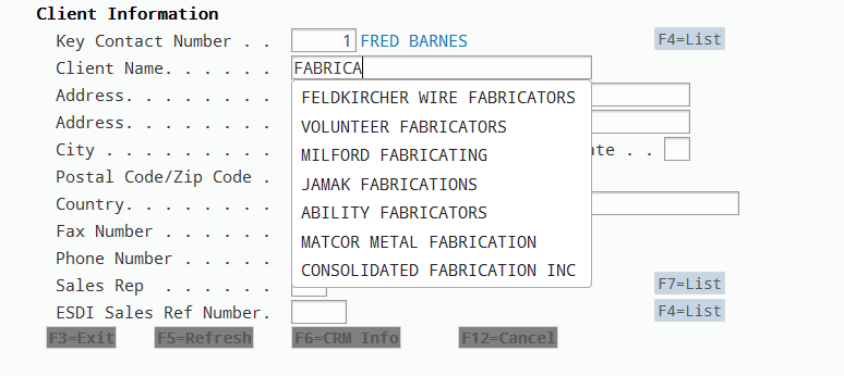


Figure 2-9. Finding a matching company for the contact.

This window is for adding contacts to a CRM system. To find the company this individual belongs to we have begun typing 'FABRICA...'. After typing the first character, the program retrieved the full contents of the Client File (about 10,000 records in this case). Following that, JavaScript subsets the list to records that contain - in any position - the characters we typed.

The JavaScript search is case-insensitive.

Using Qualified SQL Queries to Load Records Matching in INITIAL Positions

In the first example above the SQL initially loaded the entire matching set of records, then used JavaScript to limit to matches of the entered characters in any position within the database field. You can also structure your SQL such that

- a new query is executed on the server every time you type a character,
- the search is case sensitive
- the search limits to records where the entered characters are found only in the initial positions

The next section illustrates this.

Consider this example:

```
select CLCLNA from LG#LIB/LG#CLTL2 where upper(CLCLNA) like upper(:match)
```

The use of one parameter (match) tells the Presto server component to monitor the autocomplete field for changes.

If you use a parameter you must enter the name in the visual editor properties:

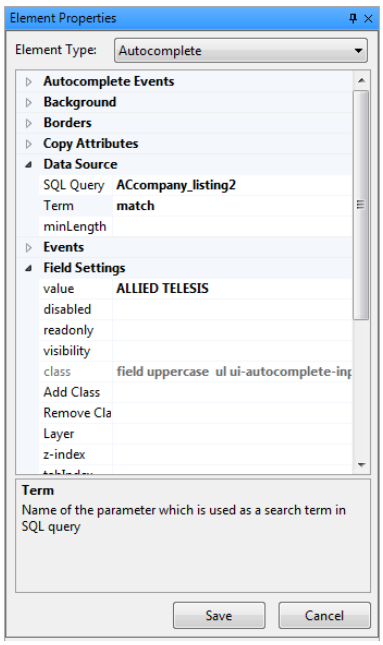


Figure 2-10. Entering the name of the search parameter passed to the query.

You can choose any name you want for the parameter.

This is the result of using the query above, with a parameter:

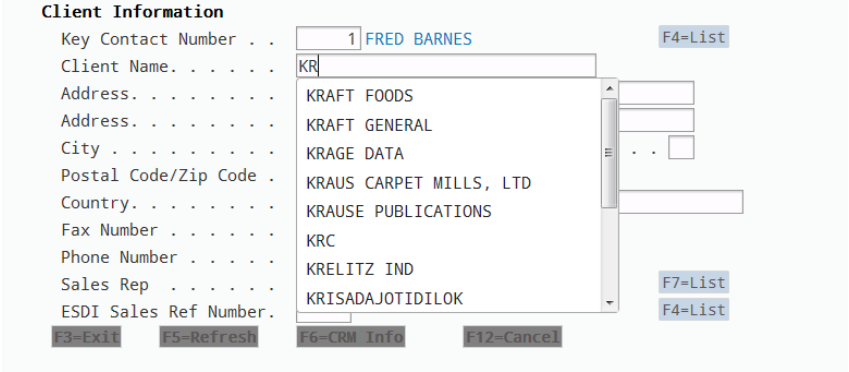


Figure 2-11. Listing records where only the initial position matches.

In this illustration we have entered the characters 'KR' for the Client Name field and the autocomplete is listing all records in the database starting with that initial position.



Note that in the query above we have use the 'upper' operation to make our search case-insensitive. The 'like' operation is case sensitive in DB2.

If in testing an autocomplete you find that in some cases it is not activated, it is possible that's happening because the value submitted from the web page does not match the case of any records in the database file.

Listing Matches in Another Order

Another nuance you can add to your SQL is to list matches in a particular order, by adding an 'order by' clause to your query. For example:

```
select CLCLNA from LG#LIB/LG#CLTL2 where upper(CLCLNA) like upper(:match) order by
CLCLNA
```

This query results in this outcome:

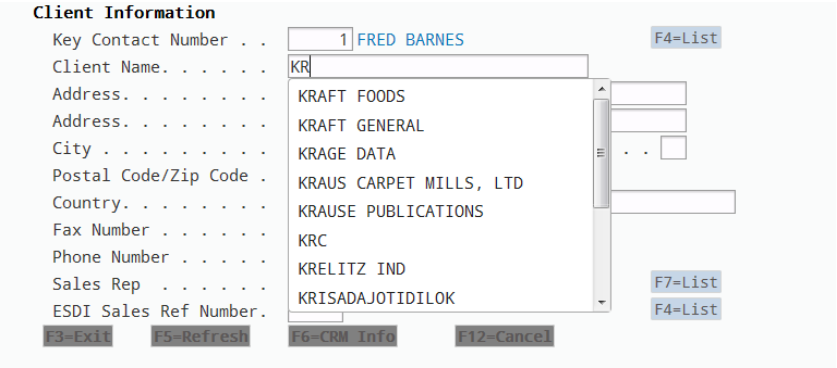


Figure 2-12. Results benefitting from an added 'order by' clause.

You can order your results in the way that will be most intuitive to your users.

Returning Matches in Any Position

You can also adjust your SQL such that it finds matches on the entered characters in any position, using a query similar to this one:

```
select CLCLNA from LG#LIB/LG#CLTL2 where upper(CLCLNA) like upper('%:match%') order by
CLCLNA
```

This will list a set of records similar to what you'd see if you loaded the entire file with an unqualified query, and then allowed the JavaScript to take over and subset the list.

Adding Additional Parameters to your SQL

You can also set up SQL queries that take additional parameters from the screen. The first example in this section uses this technique, and the section "Adding Autocomplete Functionality to Input Fields" in the online tutorial has illustrated, step by step instructions on how to use additional parameters in an autocomplete query.

jQuery Autocomplete Events

The jQuery autocomplete widget supports a number of useful events. Presto includes these events in the visual editor:

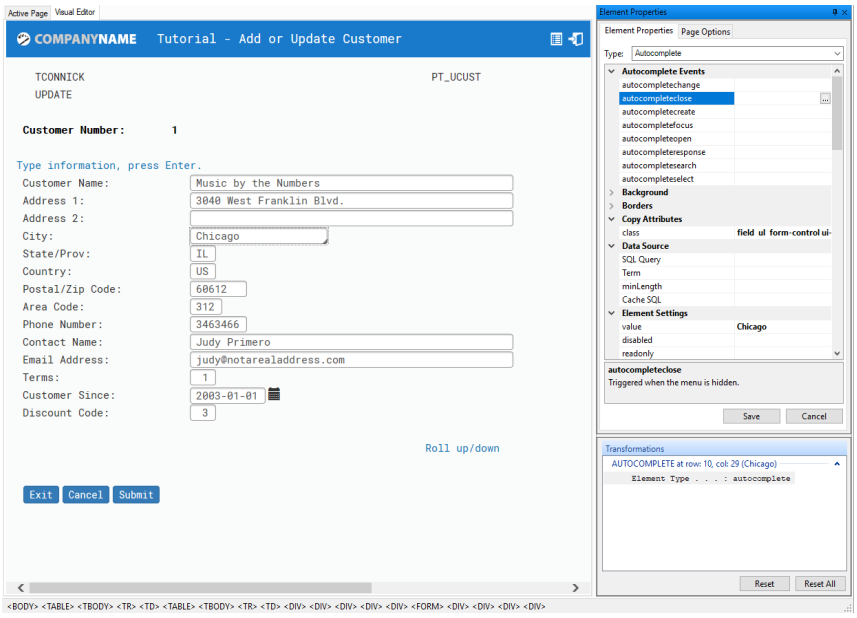


Figure 2-13. Visual editor access to the jQuery autocomplete events.

You can find complete documentation for the jQuery autocomplete widget at http://api.jqueryui.com/autocomplete/.

Block Container (div)

Use this element type to identify and tag a significant division within your page. Usually creating a div is just a precursor to adding further attributes to it.



Note that divs, like most other elements you can use Presto, are standard web application elements and objects. This means you can find more complete documentation for almost all of these online, as well as myriad examples. A popular site is http://www.w3schools.com/

A common use of block containers is to visually separate areas of a page by adding different background colors. Here is an example where a customer maintenance screen has been re-organized into two sections with distinct backgrounds:

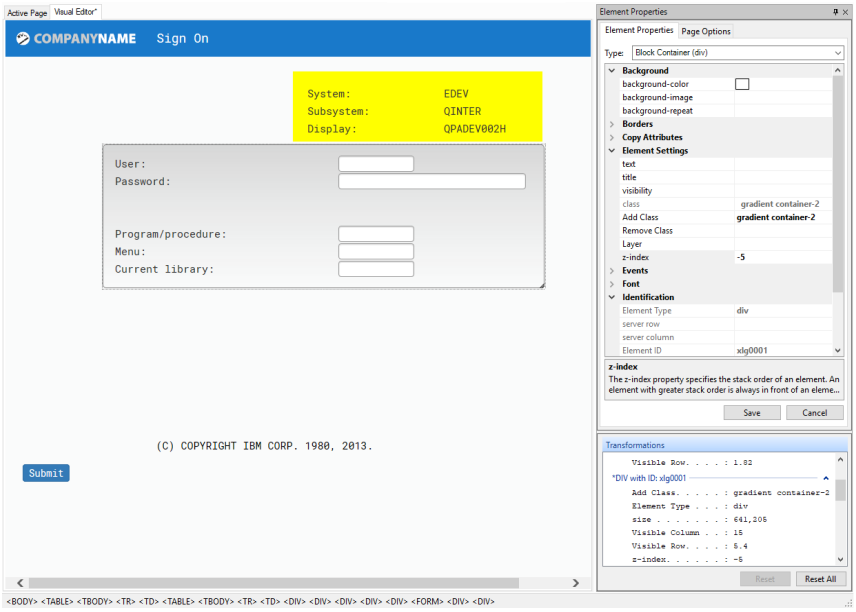


Figure 2-14. A panel re-arranged to include two Block Containers (divs) as backgrounds.

For this illustration we set the z-index (Field Settings) of the blocks to -1 to make sure they appeared under the fields and text already on the screen.

This is the simplest of implementations, but essentially you can use blocks in any situation where you need to visually separate one section of your screen from another.

Adding Block Containers and Using the layer_N Class

A useful feature when customizing complex screens with lots of fields and information is the ability to add background colors to specific areas of the screen. Consider this page:

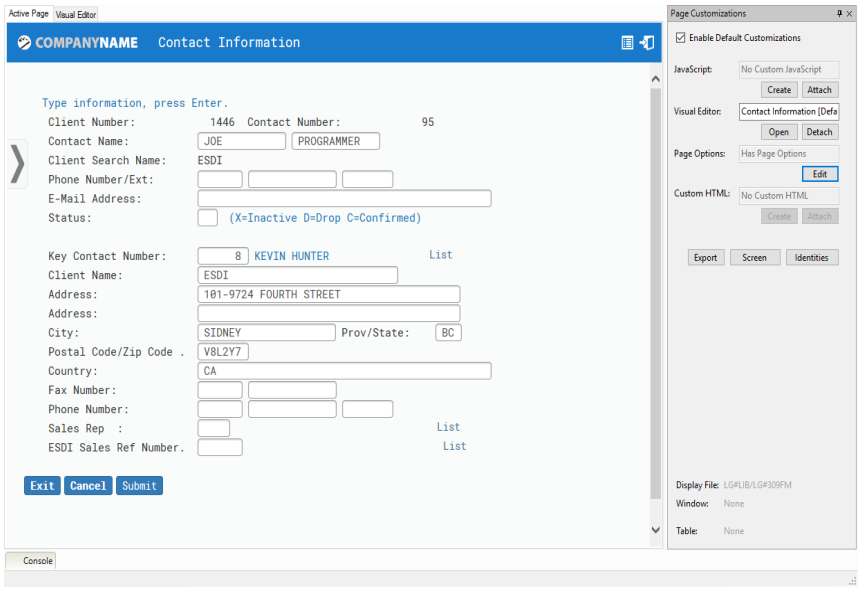


Figure 2-15. A relatively busy screen, with many fields belonging to two different categories.

The fields at the top of the screen allow us to maintain a contact record, while the ones at the bottom are for working with the company that person works for.

With the palette you can easily drag a Block Container onto the page:

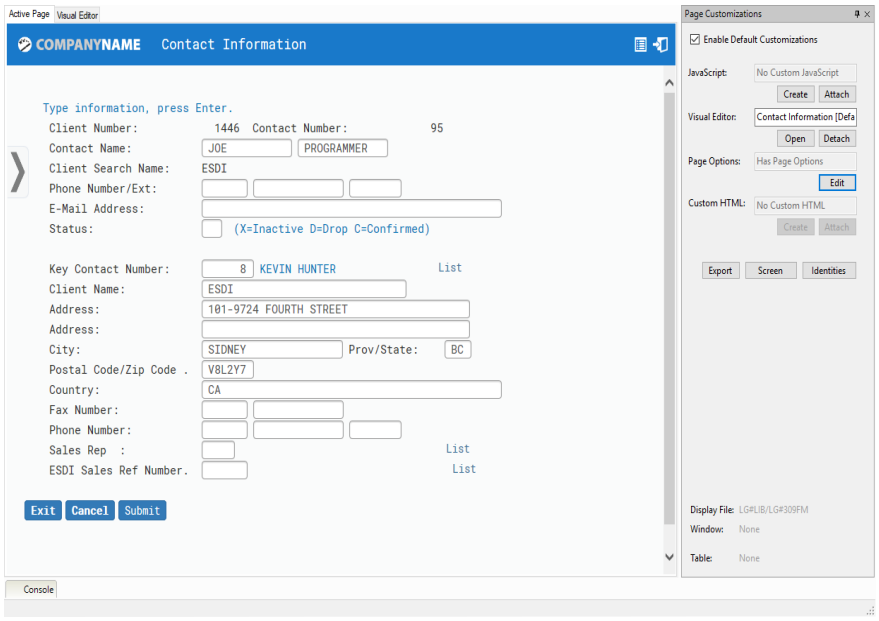


Figure 2-16. Dragging a new block container onto the page.

In this illustration we have also subsequently resized it to Then you can resize it by dragging out the bottom right hand corner:

As the next step you can set a background color for it. You can do this from the Element Properties, by finding the background-color property and selecting a color.

The outcome may not be what you expect:

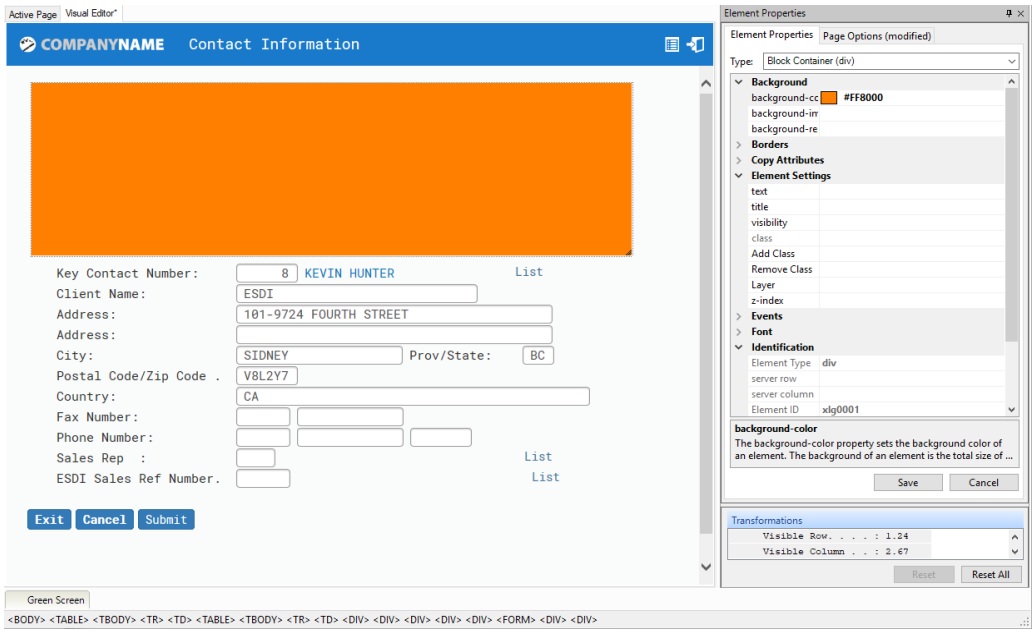


Figure 2-17. Adding an orange background to the page.

When Presto first builds a screen the default z-index for all elements is 0. The z-index is a CSS property that sets the stack order of specific elements. An element with greater stack order is always in front of another element with lower stack order.[1]

Presto ships with 11 preset classes that allow you set layer levels for various components. These classes are named layer_N, where N is a value from 0 to 10. Where N=0, the assigned z-index is 0, for N=1 the assigned z-index is -10, and so on, down to -100. If you're familiar with CSS, this is what the relevant section looks like:

```
/* Base - Default everything to 0 z-index */
#m * { z-index: 0; }

/* Layers below base layer */
#m .layer_-10 { z-index: -100; }
#m .layer_-9 { z-index: -90; }
etc...
m .layer_-1 { z-index: -10; }
#m .layer_0
{
z-index: 0;
}

/* Layers above base layer */
#m .layer_1 { z-index: 10; }
#m .layer_2 { z-index: 20; }
etc...
#m .layer_10 { z-index: 100; }
```

Here is the outcome after adding a class of 'layer_-1' to the orange container:

[1] http://en.wikipedia.org/wiki/Z-index, http://www.w3.org/TR/CSS2/visuren.html#z-index

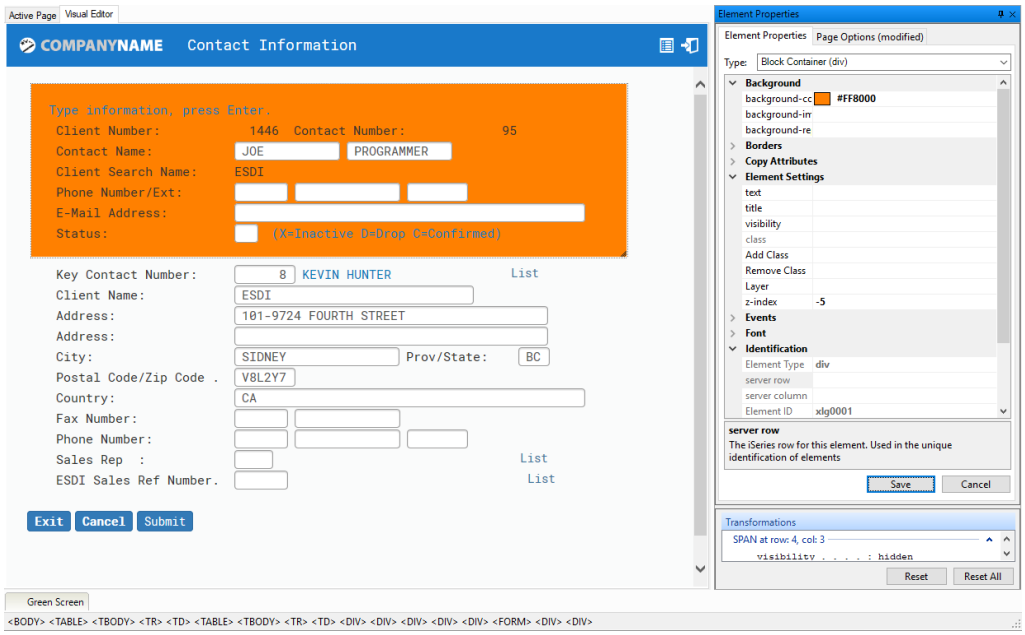


Figure 2-18. Adding a layer_1 class to our block container.

Using these classes you can easily build pages with multiple overlapping block containers.

By using block containers and layers you can easily draw your users' attention to different elements or screen areas, as well as group fields and text into logical areas, taking advantage of the additional screen real estate that you have in a web application.

Button

Use this element type to convert text to buttons, or create new buttons on your pages. Buttons are particularly useful in conjunction with events.

For example, you could create a new element on the screen, set to be of type button, then attach something like this via the Event Editor:

```
jQ('.pagedown').click(function(){SendRequest('RU');});
```

This creates a Page Down button.

Contact Technical Support or search our web site for other ideas!

Function Key

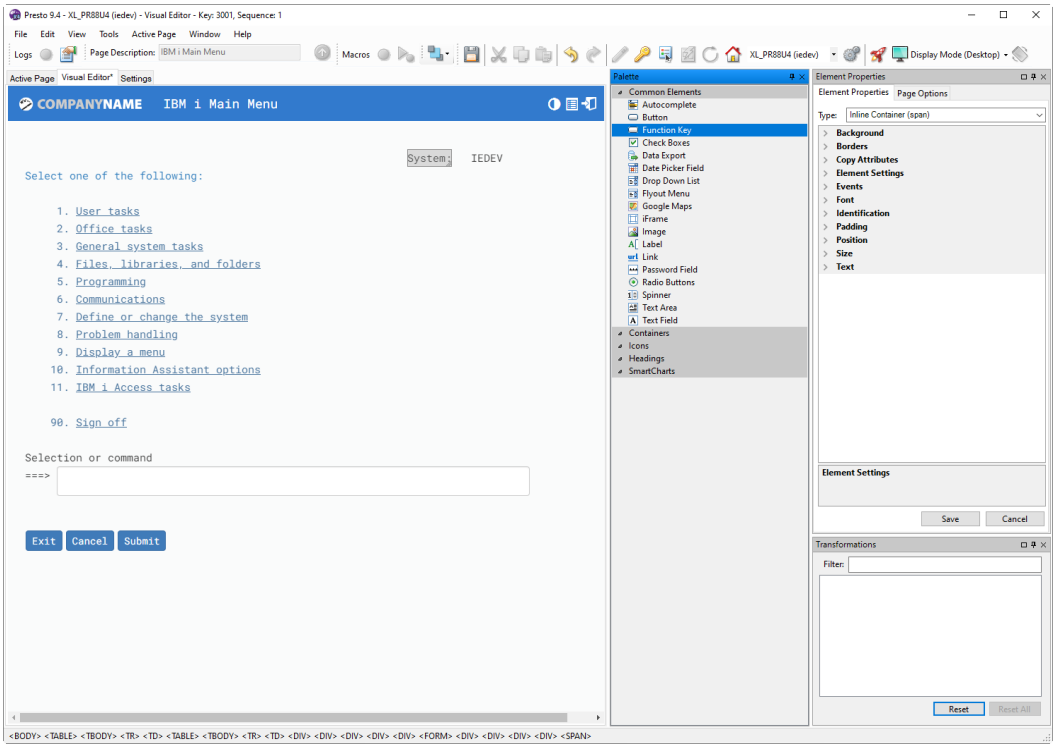
This element type adds a custom function key to the function-keys menu.

This is useful for these scenarios:

- A screen of the IBM i application uses a toggle to show additional function keys. Instead of a toggle, Presto can show all the available function keys on the screen at the same time.
- A screen has a function key that is 'hidden'- that is, the function key can be used but is not indicated on the screen. It can be added to the screen using the function-key element.
- You want a custom function key to perform some action on the screen that isn't a command, such as running JavaScript.

Adding a Function Key

You can add a Function Key to your screen using the Visual Editor right-click menu or from the palette, as shown here:



To add function keys, drag & drop the Function Key element anywhere on the screen or right-click in the function-key menu and select the 'Add Function Key' option:

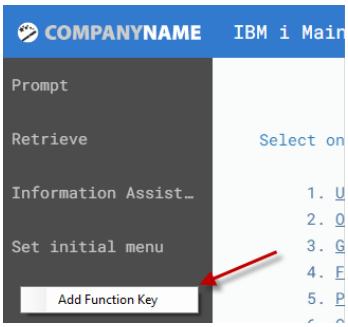
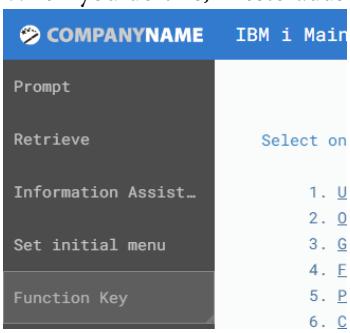


Figure 2-19. Right-clicking the sidebar to add a new function key.

When you do this, Presto adds a new function key to the sidebar:



The behavior of adding a new function changes based on the position of the function-key menu (set in Page Options):

- **Left or Right.** Presto adds a new function key to the bottom of the sidebar, and it cannot be re-arranged. The vrow/vcol are recorded from the mouse position of the right-click to use if the position of the function keys is moved to the bottom of the page.
- **Bottom**.
  - Presto allows you to drag & drop the Function Key element onto the screen same as other elements.
  - Right-clicking on a function key shows the option 'Add Function Key".
  - If the position of the function-key menu changes from sidebar to bottom, the position of custom function keys that you've already added will be based on the mouse position of where you right-clicked to add the function key.



If the position of the function-key menu changes (via Page Options), the position of custom function keys may need to be manually re-adjusted.

Function Key Field Properties

After you have added the function-key element, you can set the properties for the command it triggers and the name.

- **Function Key.** The action associated with this function. For example if you select 13 and end result is a function key that works like your keyboard F13.
- **Value.** The name displayed for the function key on the screen.

Here is the element properties displayed when a function key is selected.

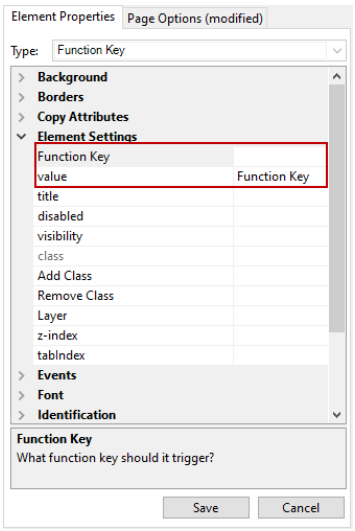


Figure 2-20. Element properties to select command and change name of Function Key.

Hiding a Function Key

You can delete the function key by selecting a function key and pressing the Delete key. This changes the visibility property to hidden.

Check Boxes

The checkbox transformation is useful for situations where you need to submit one or more selected values. The functionality is similar to the Radio transformation (documented in detail below), with the difference that multiple items can be selected.



Unchecked values return nothing to the server.

Data Export

The data export transformation allows you to add a button, link or image to your page which, when clicked, will prompt the user to download a CSV file with a set of records defined by an associated SQL statement.

For example, if you have subfile programs that lists customers, order details or part numbers you can use this feature to allow your users to quickly download a summary list of all matching records. They can then open the CSV file in a spreadsheet application to review the results or add further calculations.

Adding the Element to the Screen

You can convert an existing screen element to this type, or insert a new Export Data element on your page. You can add a Data Export button to your screen using the Visual Editor right-click menu or from the palette, as shown here:

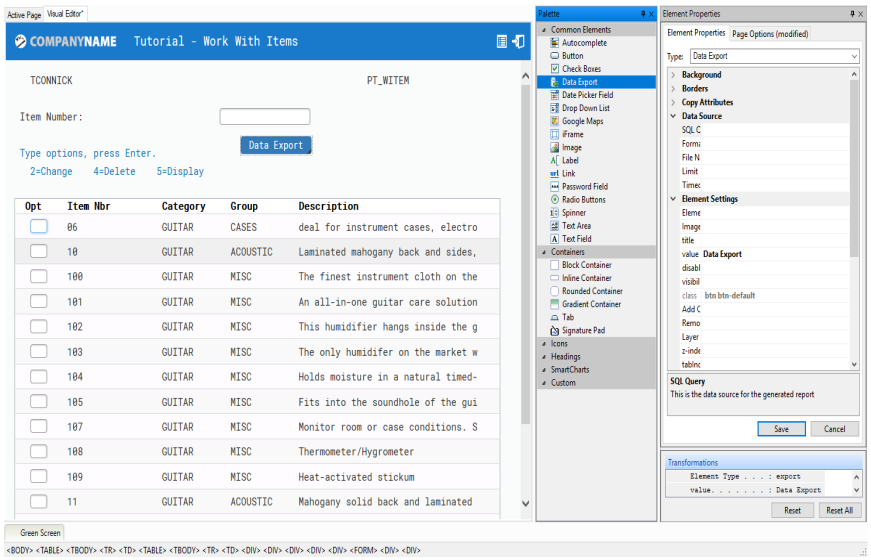


Figure 2-21. Inserting a new 'Export Data' element on the screen.

The new element defaults to being displayed as a button. Let's look at the available options.

Data Export Field Properties

After inserting the element you can adjust the properties. A number of properties are generic and shared with elements of many other classes, but let's look at the unique ones for the Export Data element.

Field Settings: Element Style

You will normally add a visible object to the screen which the user can click to initiate the action of downloading the data file. The three supported Element Styles are:

- **Button.** This is the default. If you add a button, you should also change the value property, which defaults to the generic 'Data Export'.

- **Link.** You can also allow the user to download a file via a hyperlink. Again, if you use this option you should have a meaningful value property in your Field Settings.
- **Image.** Finally, you can also prompt the user via an image. If you use an image you need to click next to the Image Source property to browse for an image to use.

Presto will only use the image if you have specified 'image' as the Export Data Element Style.

Data Source Properties

The Data Export element has three properties related to the data source: the SQL Query, the format and the file name. We'll look at the SQL query in some detail below the format and file name explanations.

Data Source: Format

You can download your data in CSV or HTML formats. HTML is by default displayed in a web page, while with CSV files the user is prompted to download the file.

Here is an illustration of the output from a simple HTML data export, listing warehouses:

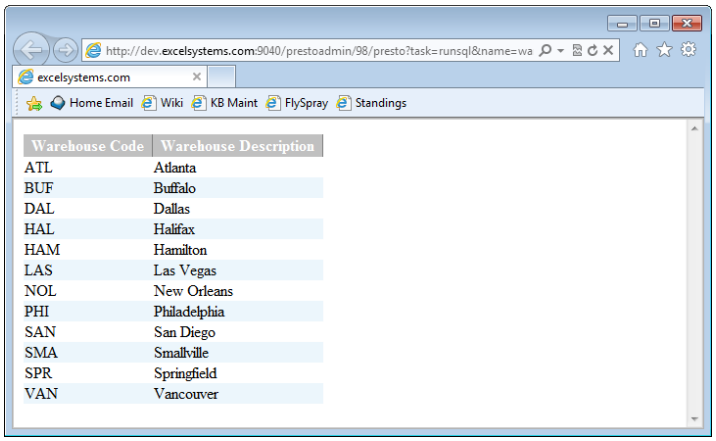


Figure 2-22. A simple HTML data export.

This was generated by attaching this SQL query to the button:

```
select MWWHCD as "Warehouse Code", MWWHDS as "Warehouse Description" from xl_prdemo/MU_
WHSF order by MWWHDS
```

You can adjust your SQL to meet the needs of the page or application to which you are outputting the data.

Data Source: File Name

This option is relevant if you select csv as your download format.

Type a default name for the downloaded file. You can use static text, screen values, or a combination of both. For example: CustomerOrders, {screen(2,14,45)}, or Customer{screen (2,14,5)}Orders.



Remember that you can drag values up from the green screen view to the properties. If you plan to combine screen values with static text this way, drag the value up first, and then add the static text, otherwise the value you drag up will overwrite what you've typed there.

The default CSV file name is the Data Source SQL query name you use with the element.

Defining the Data Source SQL

A critical property of an Export Data element is the data source SQL Query. Let's look at a simple example incorporating some screen data.

Here is a page from our tutorial application, listing the orders for one of our customers:

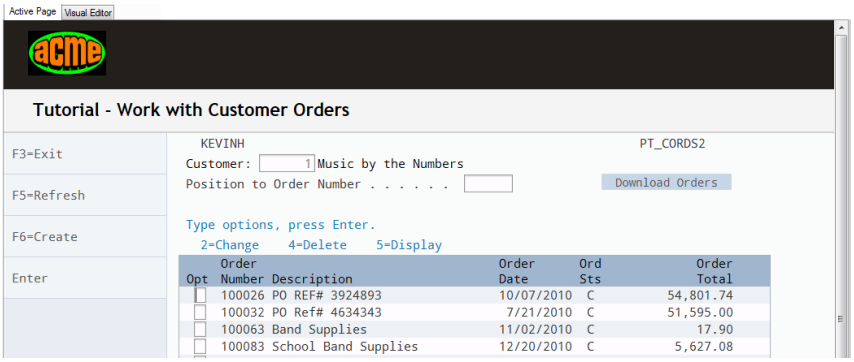


Figure 2-23. Subfile list screen with Export Data button added in the header area.

As you can see we've added a new 'Download Orders' button to the header area of this screen.

This button is configured as an HTML download with this SQL query attached to it as the data source:

```
select
MU_ORDHF.OHCUST as "Customer Number",
MU_ORDDF.ODORD as "Order Number",
MU_ORDHF.OHDESC as "Order Description",
MU_ORDDF.ODLINE as "Line #",
MU_ORDDF.ODITEM as "Item Number",
MU_ORDDF.ODPRIC as "Item Price",
MU_ORDDF.ODDISC as "Discount Percentage",
MU_ORDDF.ODQTY as "Quantity Ordered",
MU_ORDDF.ODEXTAM as "Extended Amount"
from MU_ORDDF inner join MU_ORDHF on
MU_ORDDF.ODORD = MU_ORDHF.OHORD where OHCUST = cast(:cstnbr as int)
order by MU_ORDDF.ODORD
```

Notice this bit in the second to last line:

```
where OHCUST = cast(:cstnbr as int)
```

We are pulling a section of the screen in as a parameter and then converting that to an integer. In Presto, this is how the parameter is set up:

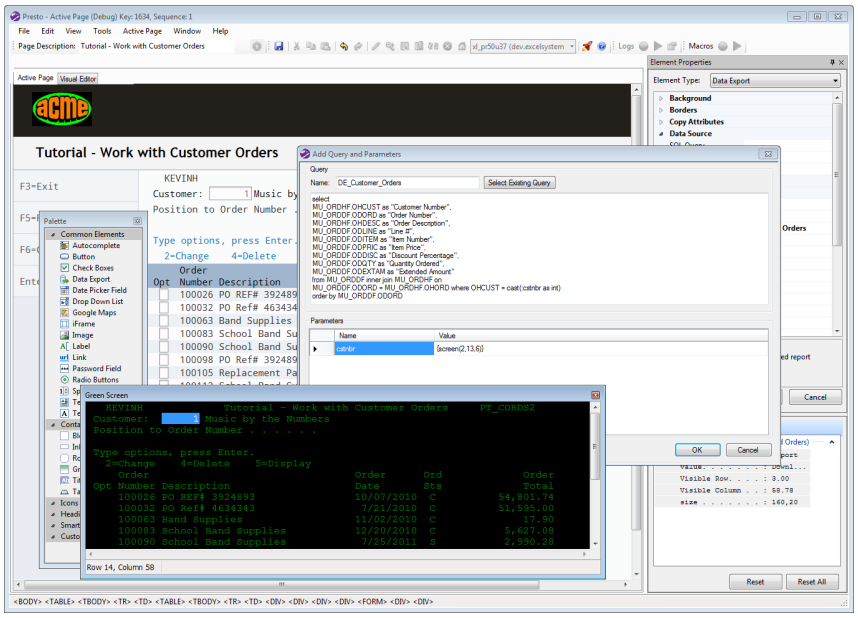
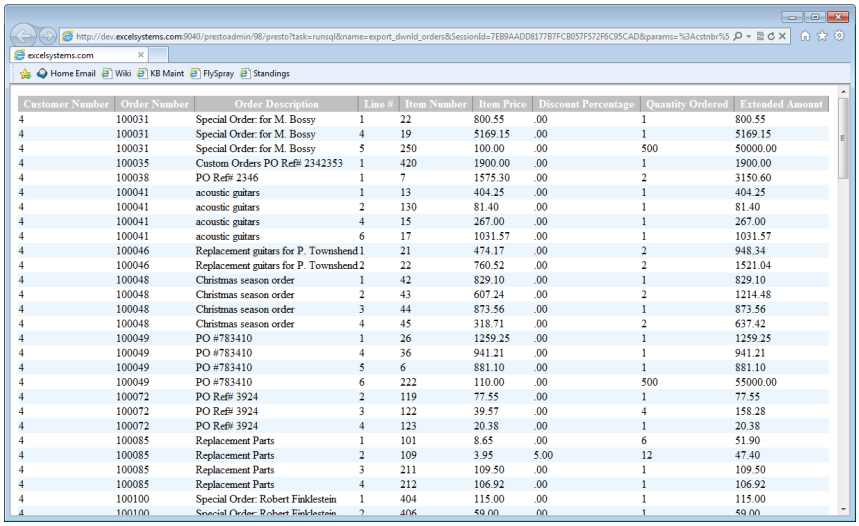


Figure 2-24. Setting up the SQL with a parm drawn from the screen.

In this illustration we highlighted the Customer Number in the green screen view and dragged it up to the Parameter Value field. The parameter name is 'cstnbr', as you can see in the SQL above.

When the user clicks on the Download Orders button in the program above (Figure 2-23), the following data is output to the browser in a new tab:



Customer Number	Order Number	Order Description	Line #	Item Number	Item Price	Discount Percentage	Quantity Ordered	Extended Amount
4	100031	Special Order: for M. Bossy	1	22	800.55	.00	1	800.55
4	100031	Special Order: for M. Bossy	4	19	5169.15	.00	1	5169.15
4	100031	Special Order: for M. Bossy	5	250	100.00	.00	500	50000.00
4	100035	Custom Orders PO Ref# 2342353	1	420	1900.00	.00	1	1900.00
4	100038	PO Ref# 2346	1	7	1575.30	.00	2	3150.60
4	100041	acoustic guitars	1	13	404.25	.00	1	404.25
4	100041	acoustic guitars	2	130	81.40	.00	1	81.40
4	100041	acoustic guitars	4	15	267.00	.00	1	267.00
4	100041	acoustic guitars	6	17	1031.57	.00	1	1031.57
4	100046	Replacement guitars for P. Townshend	1	21	474.17	.00	2	948.34
4	100046	Replacement guitars for P. Townshend	2	22	760.52	.00	2	1521.04
4	100048	Christmas season order	1	42	829.10	.00	1	829.10
4	100048	Christmas season order	2	43	607.24	.00	2	1214.48
4	100048	Christmas season order	3	44	873.56	.00	1	873.56
4	100048	Christmas season order	4	45	318.71	.00	2	637.42
4	100049	PO #783410	1	26	1259.25	.00	1	1259.25
4	100049	PO #783410	4	36	941.21	.00	1	941.21
4	100049	PO #783410	5	6	881.10	.00	1	881.10
4	100049	PO #783410	6	222	110.00	.00	500	55000.00
4	100072	PO Ref# 3924	2	119	77.55	.00	1	77.55
4	100072	PO Ref# 3924	3	122	39.57	.00	4	158.28
4	100072	PO Ref# 3924	4	123	20.38	.00	1	20.38
4	100085	Replacement Parts	1	101	8.65	.00	6	51.90
4	100085	Replacement Parts	2	109	3.95	5.00	12	47.40
4	100085	Replacement Parts	3	211	109.50	.00	1	109.50
4	100085	Replacement Parts	4	212	106.92	.00	1	106.92
4	100100	Special Order: Robert Finklestein	1	404	115.00	.00	1	115.00

Figure 2-25. Data output from the query attached to the Export Data button.

You can work with the formatting of this data using Presto's style sheet for generated tables. Contact Technical Support if you would like assistance with setting this up.

Date Picker Field

Use the date picker element to add date pickers to date input fields.

Consider the panel below, as displayed in Presto's Active Page mode:

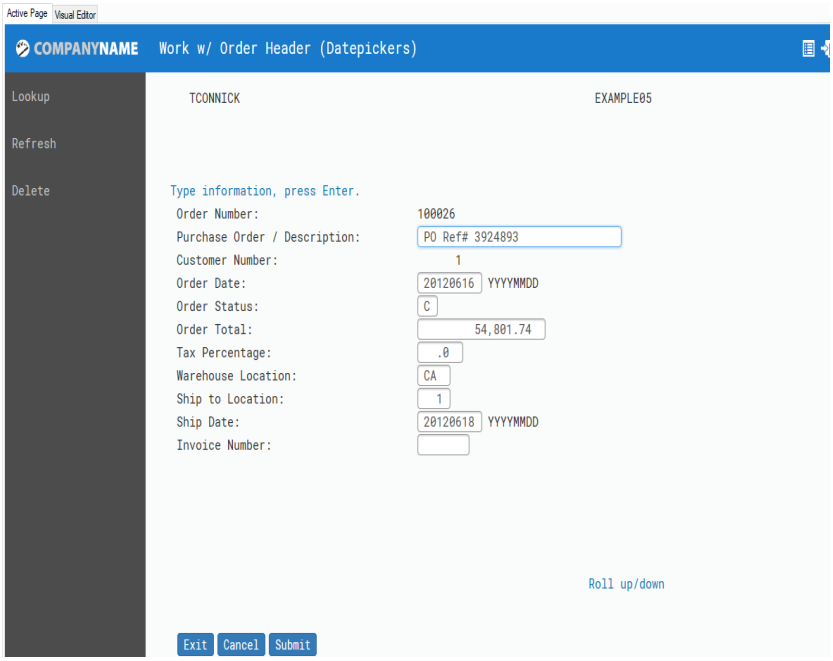


Figure 2-26. Flat panel with two date fields.

Notice that this file (our order header file MU_ORDHL1) includes both Order Ship Dates, which the user must type in YYYYMMDD format. A user-friendly enhancement to this page would be to allow the user to click the dates in a pop-up calendar and not have to worry about typing them in the correct format, or accidentally mis-typing a character.

Switching date fields to date pickers is easy. In Presto's Visual Editor click each of the dates and switch the element type from Text Field to Date Picker Field, as shown here:

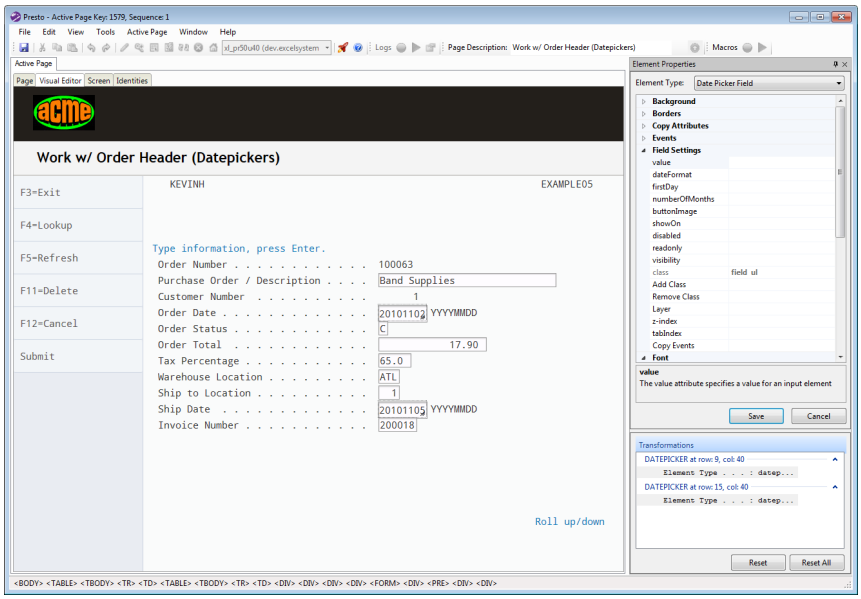


Figure 2-27. Adding a date picker to a field.



You can also switch the element type using the visual editor's right-click option - just mouse over the element and right-click to select the Transform To option.

To add a date picker to a field follow these steps:

1. Click the field.
2. In the Element Type drop down list, select Date Picker Field.
3. Save your changes.

You can review the details of the transformations as well.

When your users run this program in a browser, the date picker will look something like this:

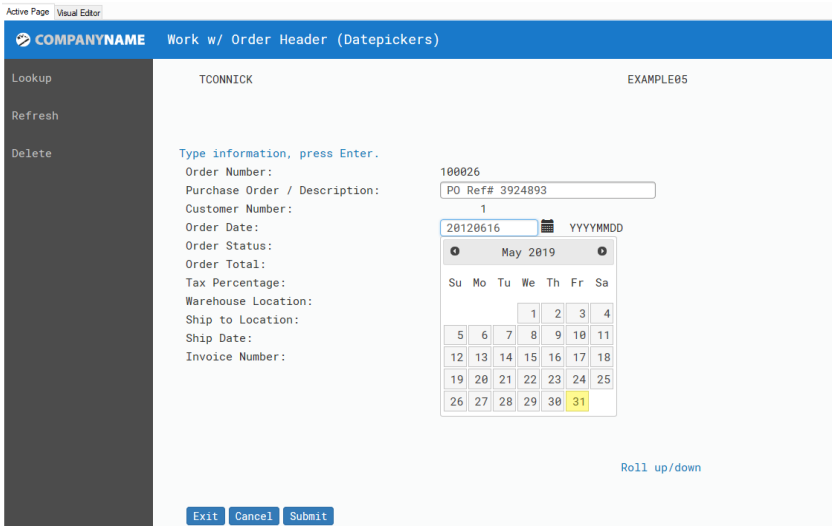


Figure 2-28. Selecting a date from a date picker.

The section "Adding Features to Your Datepickers" on page 291 includes instructions for adding several other types of date picker, and date picker options, via the Custom JavaScript interface.

Set Datepicker Options Based on Browser Language

This feature dynamically sets the datepicker language value by retrieving the language from the browser at runtime. Presto initially tries to load the language settings from userresources and if the language is not found then from Google server (this way it can work without internet connection). We ship Presto with German, French and Spanish files.

To enable this feature, set the following option:

```
<detectbrowserlanguage>1</detectbrowserlanguage>
```

in the `<javascriptoptions>` in your advanced configuration settings (Tools>Advanced Configuration).

Note that this feature is intended for environments where you may have users with many different browser language settings. If your users are using a consistent but non-English language, you can override to the appropriate language file with the following addition to your custom.js file:

```
jQuery.getScript("userresources/jquery.ui.datepicker-fr.js",
function()
{
        jQuery.datepicker.setDefaults(jQuery.datepicker.regional[ "fr" ]);
});
```

If you need another language file you can download it from the jQuery web site

(http://jquery-ui.googlecode.com/svn/tags/latest/ui/i18n/) and add it to your

ESDI/Presto/userresources folder in the IFS.

Drop Down List

With Presto you can easily use drop down boxes to replace text fields requiring users to select from lists of options. The lists of values to select from can be static or dynamic.

The following sections describe the most important element properties.

Data Source Options

List Values

Use the List Values option to build a static list of values to display to the user. This approach is illustrated in the first two examples further below.

SQL Query

Use an SQL query to dynamically generate a list of options the user can select from. This approach is particularly powerful when you want to condition the generated list based on values available elsewhere on the screen.

For example, if customers of a certain type have only a particular set of shipping options, you could include the customer type as a parameter in your SQL and use it to constrain their shipping options.

The third example below illustrates how to construct a dynamic drop down box.

Adding a Hard Coded Value or Instructions to the Top of a Dynamic List

You can combine one or more static options with a dynamically generated list. For example, if you want to generate a dynamic list of options for the user to select from, but don't want to prejudice their initial selection you can add a static value at the top, such as "Select a shipper...".

This is described in more detail in the section "Adding Static Values to Dynamic Lists" on page 232 below.

Element Settings

You can adjust other aspects of how your drop down list works via the element settings.

Default Value

Use the default value option to set the default (initially shown) value for the list. Note that this must correspond to one of the values *returned to the program*, not the text displayed on the screen (the displayed value).

For example, you may use a dynamic query to display a list of customer names that the user can choose from, but if the program is actually returning a customer number your Default Value should be that number. Likewise, if the user is selecting a warehouse, the Default Value should be the warehouse code the program is returning rather than the text description that the user sees in the drop down list.

In the following sections we will use examples to illustrate how to use this feature.

Static Example: Returning the Displayed Value

To replace a field with a static drop down box the first steps are to jump into the Visual Editor and then switch the field type. You can switch the field type in either of two ways:

- By right-clicking the field and selecting the Transform To>Common Elements>Drop Down List option.
- By selecting the field and then switching the Element Type in the Element Properties list on the right.

When you select the new element type (Drop Down List) Presto displays the available options and attributes in the list on the right.

Build the table of values to be displayed/returned from the List Values option:

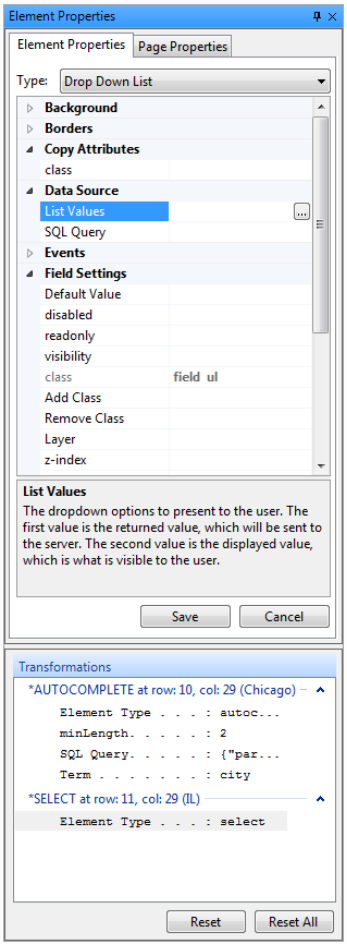


Figure 2-29. Preparing to add the options to the drop down box.

Click next to the List Values entry to present this tool:

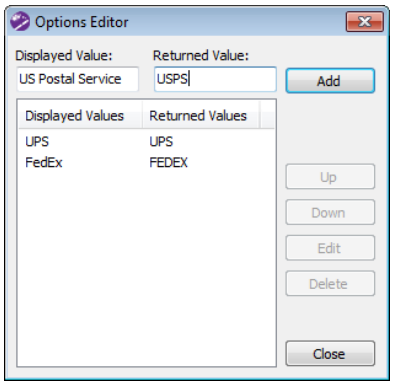


Figure 2-30. The Drop Down Box Options Editor.

Each entry must have a displayed value (which the user will see in the list) and a returned value (submitted to the program on Enter).

To add an entry type a pair of display and return values in the text fields at the top, and click the Add button (as we are about to do above). In this illustration we have added two options, and are about to add the third. Notice that we can also click on the options in the list and change the order in which they'll appear to the end user.

When you finish, save your changes. The running program will look like this:

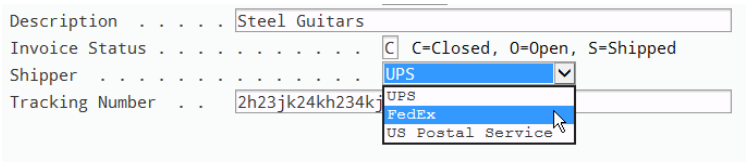


Figure 2-31. Using the new drop down box in the running program.

The list of options is available, and the program now performs some simple client side validation by not allowing the user to enter an invalid option there.



Note that this type of client side validation is useful, but to a malicious user, relatively easy to bypass. You should always use server side validation of critical fields to reinforce any validation you do in the browser.

As a final step, you may want to also hide the text that was previously to the right of the field (as we have already done above).

Static Example: Displaying a Description for a Returned Code

Another candidate for this feature is the Invoice Status field in this program. If you look at the illustration above, you can see that it also has a limited set of possible options (C, O and S). The one variation is that in this case, the program is only passing a single character field back to the file.

To address this, your drop down box options could look something like this:

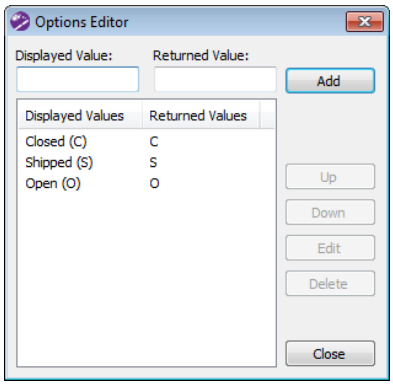


Figure 2-32. Adding options for the status field.

In this case, the Returned Value is a single digit, and to be more user-friendly, the Displayed Value includes both the code and the description.

The result in the program looks like this, after also making the text description invisible:

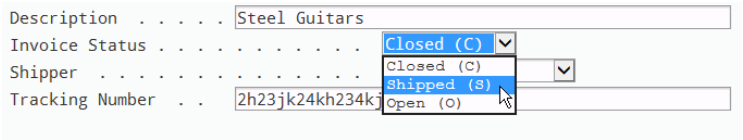


Figure 2-33. A drop down box that returns a 1,A value, but displays text in the list.

This design strategy lets your users keep their hand on their beloved mouse!



It is also possible to add entirely new drop down boxes to your pages using both the visual editor and custom scripts. The sections "Adding a New Drop Down List of Commands" on page 288 and "Using an External Web Program or Service to get Data for a Drop Down Box" on page 303 have two examples.

Dynamic Example: Using SQL Queries to Generate Lists

You can also create and use SQL queries to load drop down box values.

To add an SQL query to Presto start by selecting the Tools>SQL Queries menu option in the Presto Designer. Then click Add to create a new query:

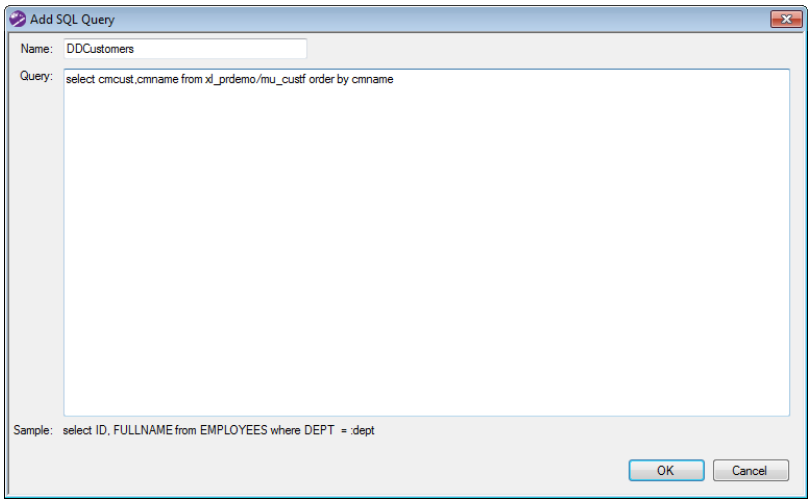


Figure 2-34. New SQL query named 'DDCustomers'.

We have named the query 'DDCustomers', and defined this SQL:

```
select cmcust,cmname from xl_prdemo/mu_custf order by cmname
```

Click OK to save your query, and close the box displaying the list.



The library reference in the query (to xl_prdemo) is optional.

The Structure of the SQL Query

The structure of the query above follows this pattern:

```
select Returned Value, Displayed Value from Library/File where Field = :Parameter order
by Field
```

The filter and ordering information are optional. So in our example above field 'cmcust' is the value returned by the selection and cmname is the value displayed in the list.

In the Visual Editor select an element (in this case, the Customer field) and change the type to Drop Down Box. Then right-click the field a second time to select SQL Query as the Data Source:

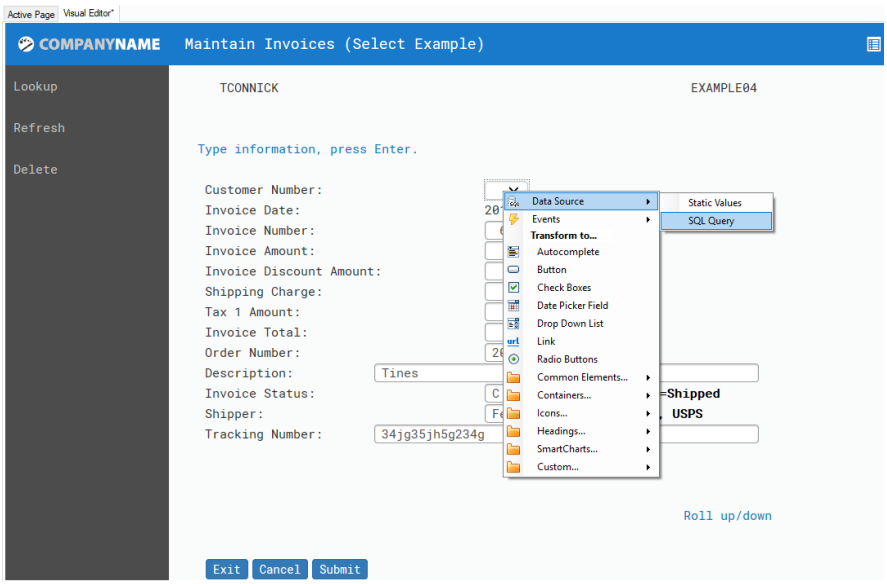


Figure 2-35. Building a dynamic drop down box.

When you select this option Presto prompts you to build a query, or select an existing one.



You can also add the query from the Element Properties menu on the right.

This is the Query Selection window (you can get to this by clicking the 'Select Existing Query' button at the Add Query and Parameters window):

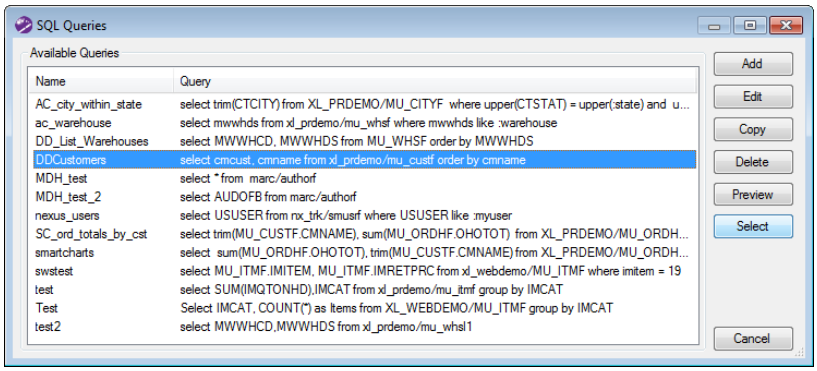


Figure 2-36. Selecting an available SQL query in the Dynamic Options Editor.

In this illustration we have several queries to choose from. To select an entry, click it in the list at the top, and click OK. The one we are using is called 'DDCustomers'.



Using Query Parameters

You can also use parameters in your query, and pull the parameter values from the screen you're working on. To use this feature select the View menu option to also display the

green screen preview below the visual editor, and drag & drop values from the green screen preview to your parameter values.

The section on autocompletes includes examples using parameters.

Here is our program at run-time:



Figure 2-37. Using the new dynamic drop down list.

In this illustration, we are in the process of creating a new invoice, and selecting the customer from the list.

The list of customers is dynamically loaded from the file when the program is run. This technique is more effective than a lookup program as the user isn't forced to leave the main screen. It may not, however, be suited for situations with more than about a dozen valid entries. In those cases you'd be better off using an autocomplete.

Adding Static Values to Dynamic Lists

You can combine one or more static options with a dynamically generated list.

For example, if you want to generate a dynamic list of options for the user to select from, but don't want to prejudice their initial selection, you can add a static text string at the top of the list with return value of zero (or another non-valid value).

Here is an illustration of how we have set up a static value to serve this purpose:

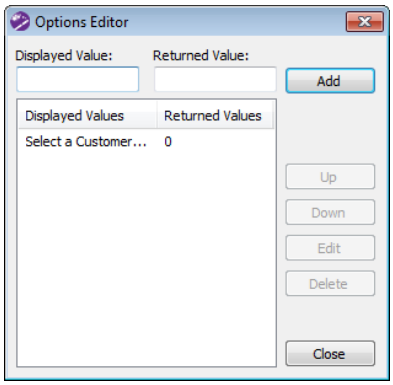


Figure 2-38. Adding a static value to the top of a dynamically generated list.

At run time, the screen would show a drop down box with an initial value of 'Select a Customer...'. The user could then select an entry from the list. If they didn't select an entry, it would be the same as selecting zero or blanks in the initial green screen program.

Blank Option for Dropdown

This new option allows you to add a blank entry at the top of a dropdown.

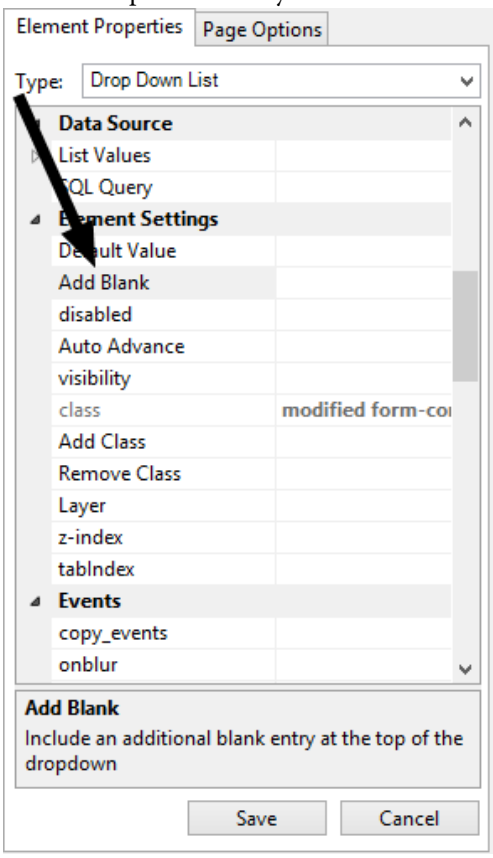


Figure 2-39. Including a blank value.

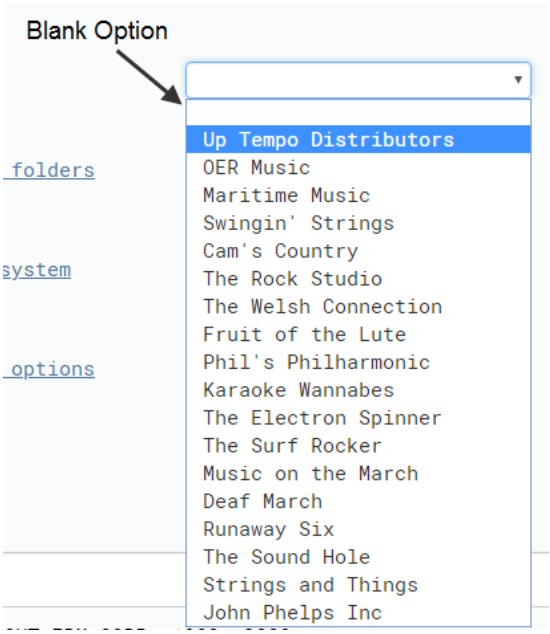


Figure 2-40. Blank entry in dropdown list.

Google® Maps

A useful enhancement to your applications is to load a Google map snapshot of an address on a page. You can use the visual editor to add a map to your page, pulling the key address information from fields on the screen.



Check our online Knowledge Base article number 15349 for instructions to add fully dynamic Google maps to your application. What works well is to embed a snapshot of an address in your original page, and then link that out to a more complete and full-featured full page map.

Here is an illustration of a screen used to update customers in the Presto tutorial application:

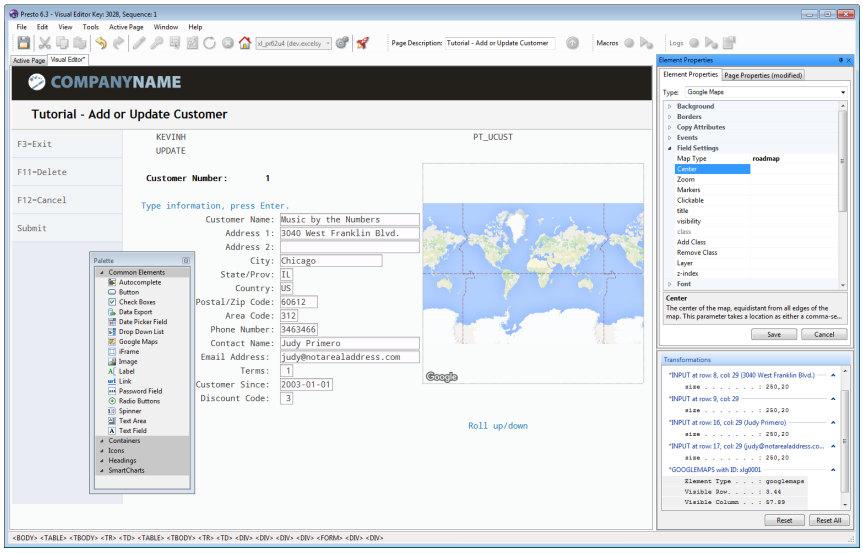


Figure 2-41. Preparing to add a Google map to a customer details page.

In this illustration we started by clicking all the longer fields on the page and shortening them up to make more room on the right side of the page. Then we dragged a Google Map from the palette into the empty space, and resized the map to better fit the available area.

Then we followed the steps below to tune the map.

Step 1: Type a sample address into the Field Settings 'Center' property.

While you are setting up your map it's simplest to start with manually entering a sample address into the Center property (Field Settings group). With this you can decide on the best map type, zoom level, and other attributes.

For our example we have entered the following value from the first customer in our tutorial application - 7844 Franklin Blvd. Champaign, IL USA - and picked an initial zoom level of 6 in a roadmap:

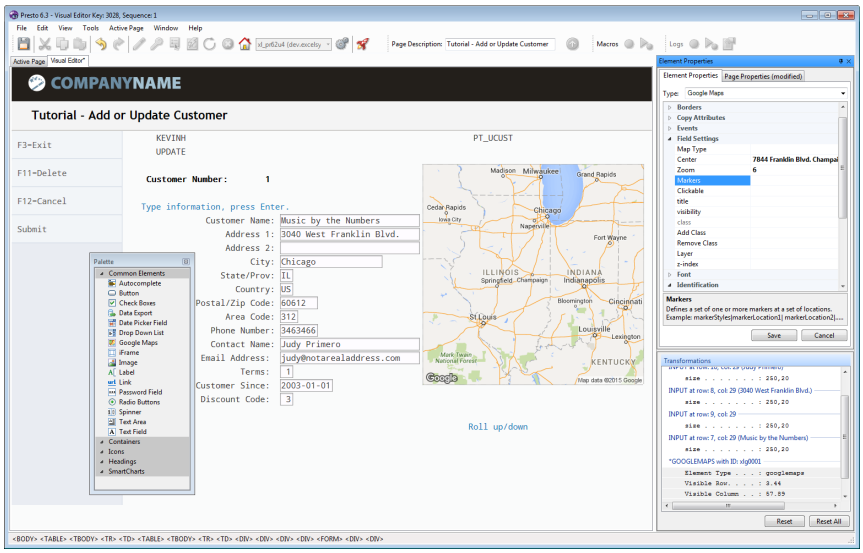


Figure 2-42. Evaluating some initial settings for a Google map.

The map is functional, but could use a few further improvements.

Step 2: Select a Map Type.

Adjust the Map Type to best meet your needs. Options are:

- **roadmap.** Loads a standard road map.
- **satellite.** Displays a satellite image of the location.
- **hybrid.** Shows a satellite image with a roadmap grid imposed on it.
- **terrain.** Shows the terrain in the area surrounding the selected address. Terrain snapshots at high zoom levels may not be useful.
- **streetview.** Shows a street-level view of the address.

The style of map best suited to your needs may also be a function of the zoom level that your users find most useful.

Step 3: Select the zoom level.

The Google map supports zoom levels from 1 to 18. Experiment with this value to find the right balance for your needs.

Step 4: Replace the static address with variables from your page.

Once you have settled on the best settings for your map, replace the hard coded value you used for testing purposes with dynamic values from the screen.

In this illustration we have opened the Green Screen view (View>Green Screen) and are in the process of dragging the State field over to the Center property, where we've already placed the Address 1 and City fields:

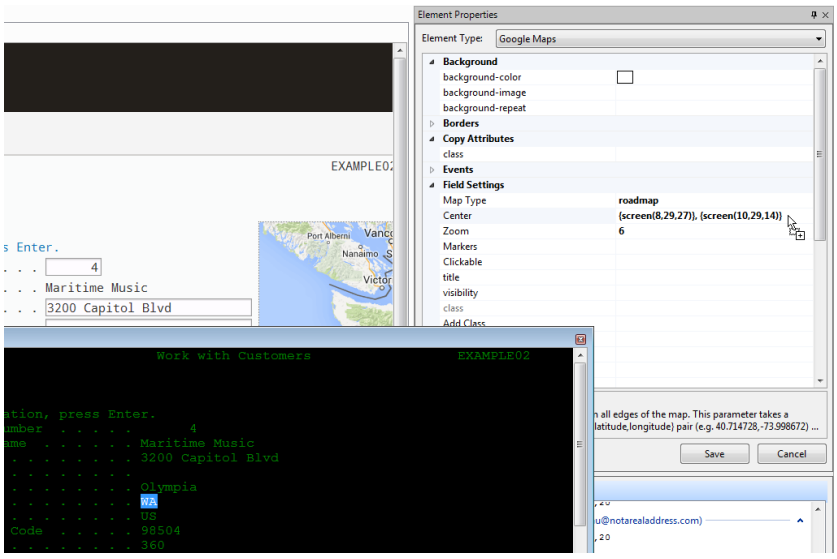


Figure 2-43. Adding screen values to the Center property.

This will be the resulting value for our Center property:

{screen(8,29,28)}, {screen(10,29,17)} {screen(11,29,2)}

You can also type this by hand rather than using the green screen preview.

Step 5: Save your changes and test!

To finish up save your changes and test your map settings with several more customers.

Adding Markers

Use the Markers property to add a marker at the center location or other nearby addresses.

In this illustration we have defined the center at our office in Sidney, BC, and added 4 markers for our favorite coffee shops in the area:

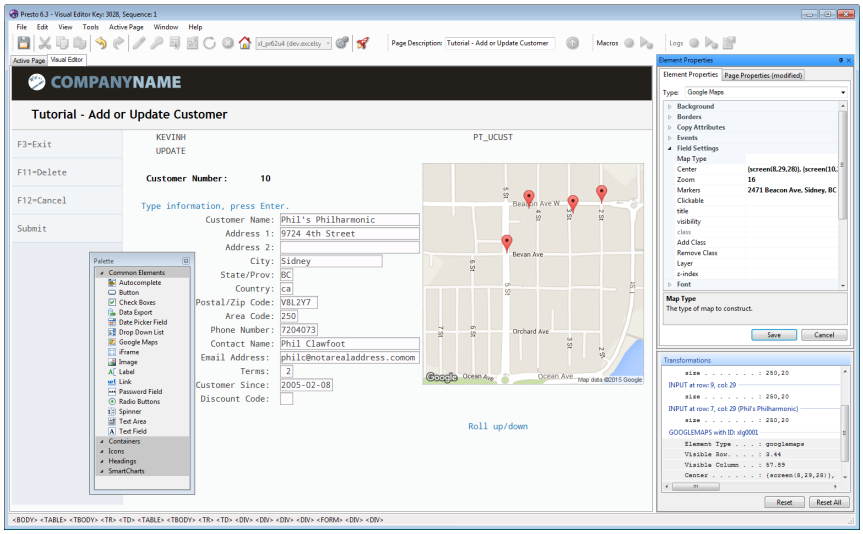


Figure 2-44. Adding multiple markers near an address.

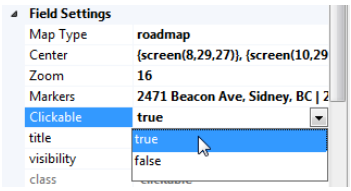
To add multiple markers separate them with a vertical line. This is the value used in the illustration above:

2471 Beacon Ave, Sidney, BC | 2400 Bevan Avenue, Sidney, BC | 2423 Beacon Avenue, Sidney, BC | 2505 Beacon Avenue, Sidney BC

Linking a Map Snapshot to a Full Featured Google Map

As mentioned above, what works well in an application is to add a small map snapshot in your page, and link the snapshot to a more full featured map that opens in a new browser tab or window.

To do this, select a value of 'true' for the Clickable property:



Icon

The visual editor palette includes a number of icons to choose from:

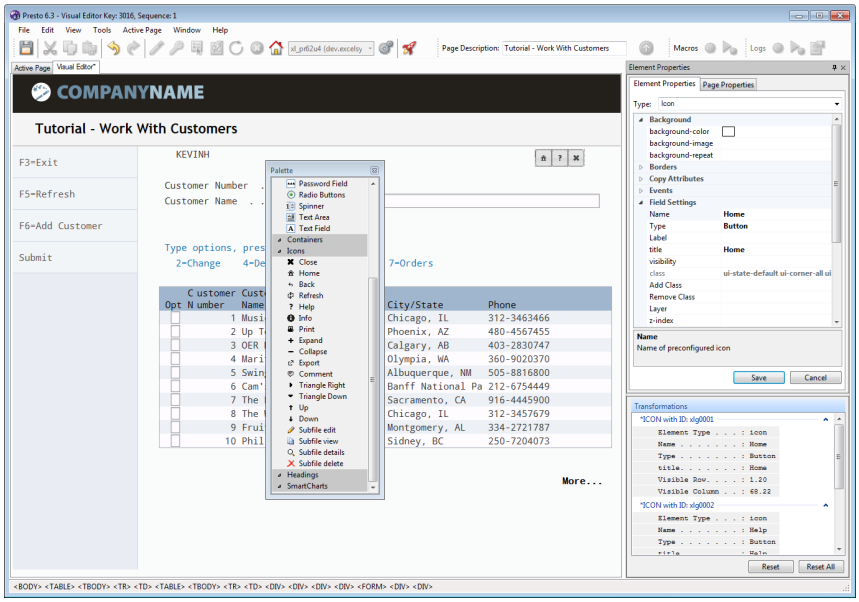


Figure 2-45. New Presto icons, to use in various ways on your screens.

The icons are available for your use from the visual editor and they are also installed in your IFS allowing you to easily use them in custom code. Note for example that the last four are intended to be used as subfile option actions.

In the illustration above we have added three icons as buttons to the top right of our screen. They can then have various actions associated with them (Help, Back or F12, Print Page, etc.).

Field Settings

The following field settings are particularly relevant to the icons you use in your pages:

- **Icon vs. Button View.** The icon Field Settings include a 'Type' property. Two types are supported: Icon and Button. Buttons have a gradient background, while Icons are transparent.
- **Label.** Use the label to display text next to the icon.
- **Title.** This value is displayed when the user hovers their mouse over the icon.

You can use icons on your screens in many ways - as clickable buttons, scroll bar arrows, links to other sites, etc.

iFrame

The iframe element is useful for inserting content in your page.



A restriction with iframes is that if the page or content you are loading is not from the same domain in which Presto is running, the loaded content cannot include JavaScript that tries to access the main Presto page, as this will trigger a security error.

If the page belongs to you, you can change “document.domain” to match Presto’s domain.

In this illustration we have inserted a new iFrame into our page and resized it to span the width of the page:

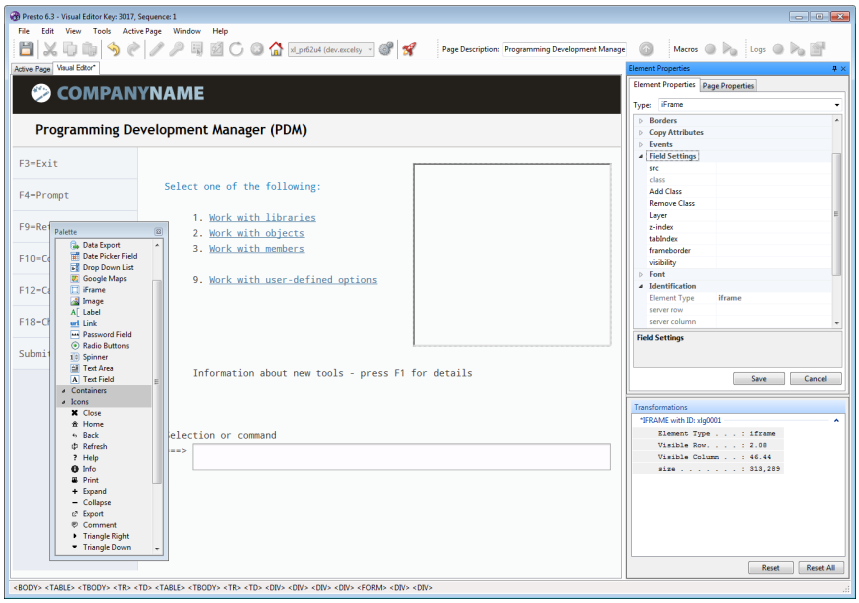


Figure 2-46. Inserting a new iFrame into your page with the visual editor.

When you add the iframe, Presto displays the relevant properties in the list on the right.

You can drag & drop the iFrame to move it, or use the bottom and right edges to resize it.

There are many uses for this feature, allowing you to add many different types of content to your pages, to enhance the functionality of the original green screen program.

Image

Use this element to add static images to your panels. This element requires an image source value (in the Field Settings area of your properties). In this illustration we have dragged an image onto the page from the palette, and are about to browse for the actual image to add to the screen:



Figure 2-47. Browsing for an image to add to the screen.

Use the Browse button to select an image. Presto then uploads it to the customimages directory on the IFS (/esdi/XL_PRESTO/).

After selecting an image our page looks like this (after also closing the palette):

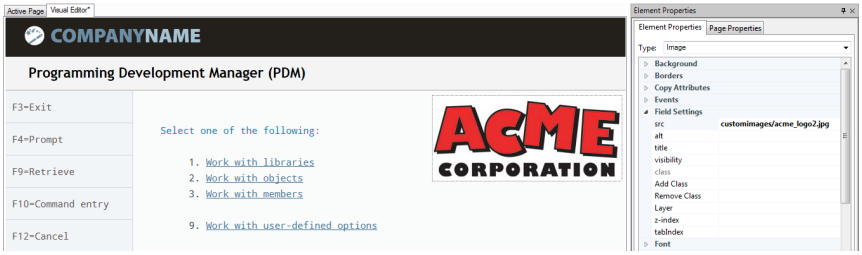


Figure 2-48. The menu program with a header image.

The tutorial includes a more detailed, step by step illustration of this technique in the section "Changing Elements to New Types" in the online tutorial.



This section describes how to add static images. The section "Using a Script to Add Dynamic Images to a Page" on page 294 describes one way to add dynamic images to your screens. The section "Using Screen Values to Dynamically Load Images" in the online tutorial describes another approach using the visual editor.

Inline Container (span)

Presto by default renders all text strings as Inline Containers (spans). Because our source is a green screen data stream, this means most things on your screen will start out tagged as spans.

You can edit many span properties, or where necessary, transform your spans into other element types. Most of these characteristics are described in , General Transformation Properties.

Label

Use the 'label' element type to associate labels with fields.

Using Labels for Consistent Styling

The main benefit of this feature is that you can use CSS to give all your labels a consistent style.

If you plan to use labels in this way you can Ctrl+Click (or use your mouse to drag a box around) all the field text descriptions on your screen and convert them to labels in one step.

Using Labels to Expand a Field's Clickable Area

You can also associate a label with specific fields to increase the clickable area for that field. This means that if a user wants to position their cursor in a field, they can click (or touch) anywhere in the area of the field or label, and the cursor will go to that input. This feature is particularly useful on mobile applications where high resolutions and small UIs may mean target areas to click are small.

To enable this feature you must add the field's Element ID to the label's 'for' property. To find the field's ID go to the Identification properties of that field and make a note of the Element ID. Generally speaking, the first field on the screen will be f0, the next will be f1, then f2, and so on.

Here we have added a 'for' value of f0 to the text label for the User on the Sign On screen:

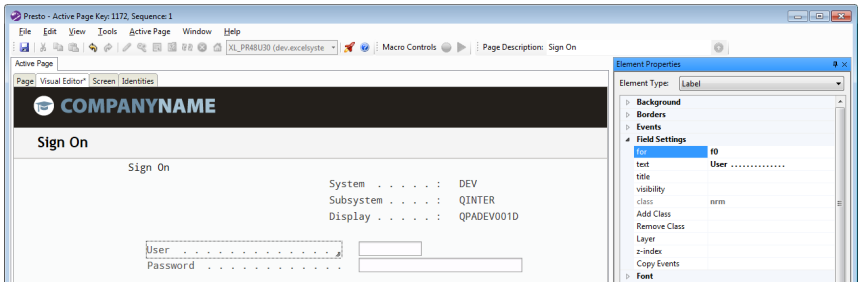


Figure 2-49. Attaching a 'for' value to the label.

The net effect of this change is that a user can click anywhere on the label or field to place the cursor in that input field.

Link

The anchor tag is useful for placing hyperlinks within your page. When you switch an element to this type you can then add an href value in the field settings. Type the URL to open.



Unless you open the link in a new tab (target '_blank') your Presto session is ended before accessing the external URL.

Password Field

You can use the password element for IBM i password fields.

Radio Buttons Transformation

You can transform a field to radio buttons using the visual editor or custom HTML. Radio buttons are particularly suited for fields with a small set of possible values, where there is room enough on the page to list all options along with an adjacent button. They are a particularly user-friendly feature as they clearly show the user their options, while also preventing non-malicious users from entering invalid values .



Never forget that client side validation can be easily circumvented by malicious users, and should always be supported by robust server side validation

Switching a Field to Radio Buttons

In our demo application the Item Master file has a field requiring one of three values:

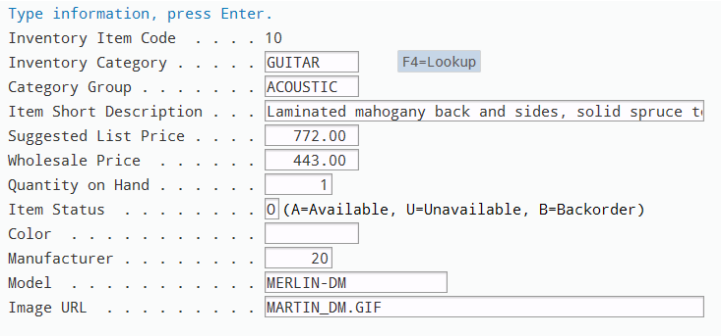


Figure 2-50. The Item Status field, requiring one of three valid values.

To transform this to Radio Buttons, click the input field and change the element type to 'Radio Buttons' in the Element Type drop down list:

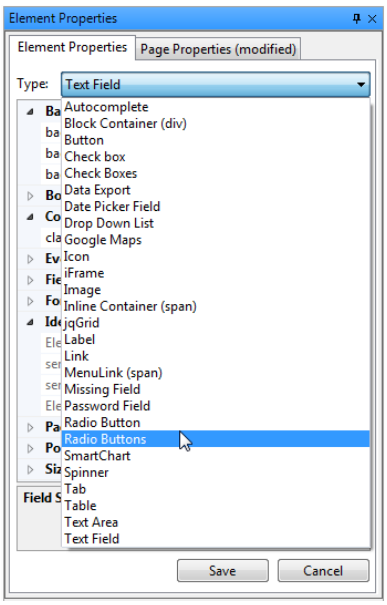


Figure 2-51. Switching an element to radio buttons.

Once you make the change above the visual editor list of properties will include the Radios Values. You can create the options using a static list of values or a dynamic list created via an SQL query. This second option is appropriate in cases where your valid values are database driven.

Creating a Static List of Options

To create a static list of options, find the Data Source properties, and click the prompt button in the 'Radio Options' property to display this prompt:

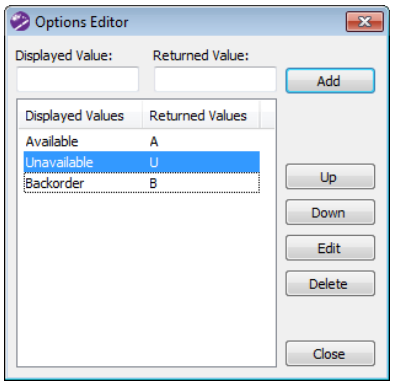


Figure 2-52. Adding static values to a radio button (via the visual editor).

To add options, enter a Displayed/Returned value pair, where the displayed value is shown on the screen and the returned value is returned to the server.

Use the Up/Down buttons to order your list entries.

When you click OK above and save your changes, this is the outcome:



Figure 2-53. Radio buttons for the Item Status code.

In this illustration we have also hidden the existing description text.

Setting Dynamic Options

If you have various 'codes' tables on your system it may be practical for you to load a list of valid options using SQL. Depending on how your subsets of valid options are established, your SQL could pick up high level keys or parameters from the screen, or you could define various static SQL queries that return the required subsets.

And of course, this option is relevant for any small list of valid options: let's consider an example using the demo files. The demo app has an option to Work with Customer Orders where each order requires the selection of a warehouse location. While this list of available warehouse is a little longer than what you'd typically use radio buttons for, that particular screen has enough room to handle it.

These are the steps to transform a field to a dynamically loaded radio buttons:

1. In the Presto Designer, navigate to the screen.
2. Click the Visual Editor tab.
3. Click the field to convert (in our case, the Warehouse Location).
4. Change the element type to Radio Buttons.
5. In the Data Source section click the SQL Query option and open the query selection window.
6. Create and attach a query. Ours is: `select MWWHCD, MWWHDS from xl_prdemo/mu_whsf order by MWWHDS.`
   The first field selected is the value returned (the Warehouse Code) and the second is the value to be displayed (the description).
7. In the Add Class property of the Field Settings, add the class "vertical-inputs" (no quotes required). This aligns the options vertically rather than horizontally.

Save your changes and review the results.

This is how the list looks:

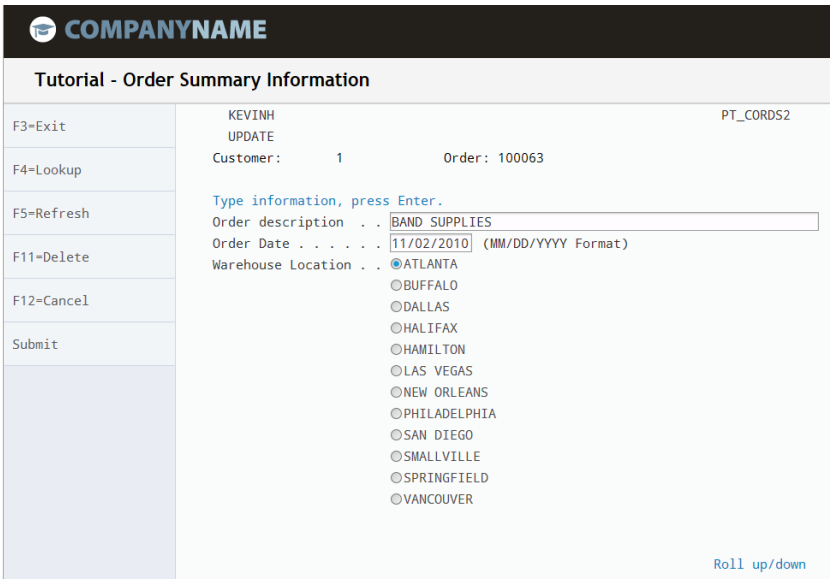


Figure 2-54. A panel where an input field has been replaced with radio buttons.

As we said before, the length of this list is outside margins of appropriate radio button use, but it illustrates the concept.

Signature Pad

Use this element to add a signature pad to your screen.

The signature images are saved to esdi/environment_name/uploads by default.

Using the Signature Pad in a Custom Skin

If you're using a custom Presto skin, you need to add these lines to your layout.html and layout_mobile.html for the signature pad to work. They can be copied to the same location as the rest of your <script> and <link> elements near the top of the file.

```
<script type="text/javascript" src="javascript/jquery.signaturepad.js?v={IFS_
VERSION}"></script>
<link type="text/css" href="javascript/css/jquery.signaturepad.css?v={IFS_VERSION}"
rel="stylesheet">
```

Adding Custom Validation and Event Handling

You can add validation and event handling to the signature pad with the following four Field Settings:

- **Save Button Text.** The text to display on the 'Save' button. Clicking this saves the current signature image to your IBM i.
- **Clear Button Text.** The text to display on the 'Clear' button. Clicking this wipes the current(unsaved) signature image.
- **Filename. Filename of the uploaded signature.**
- **Filename Target.** This is the ID of an input field on the page in which to save the filename of the generated image.
- **File Type.** File type in which signature will be saved.

Here is an illustration of the Field Settings after inserting a Signature Pad:

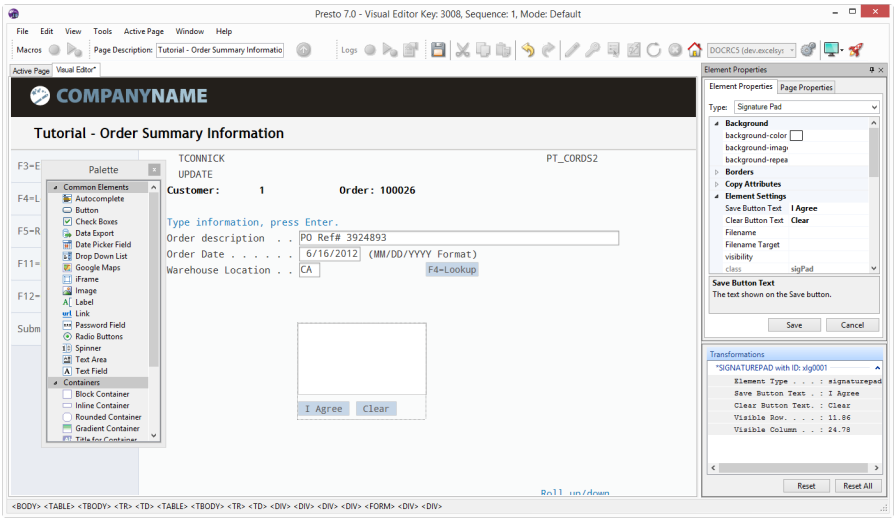


Figure 2-55. Inserting a new Signature Pad into your page with the visual editor.

There are also four new events that you can access. These are not under the typical 'Events' heading in the Element Properties tab, but in a new heading called 'Signature Save Events':

- **Uploadsuccess.** This will be triggered on successful upload of a user's signature.
- **Uploadexists.** This will be triggered if for some reason the signature file already exists.
- **Uploaderror.** This will be triggered for server-side upload errors.
- **Uploadcomplete.** This will always be triggered at the end of the signature upload.

All of these events contain this result data structure:

```
{
     "status": "<status>",
     "filename": "<filename>",
     "path": "<path>",
     "url": "<url>",
     "message": "<message>"
}
```

In some cases, events may not contain data for the structure. For example, an upload error doesn't contain a URL because there is no file created to access with a URL.

Example

You can access the result data structure from the Event with JavaScript like the following example:

```
var SigFilename = arguments[1].filename;
```

This example access the filename of the uploaded signature.

SmartCharts

SmartCharts™ is a charting component you can use to render data-driven animated charts. SmartCharts are made in JavaScript, and you can use them with any web scripting language like PML, HTML, .NET, ASP, JPS, PHP, etc., to deliver interactive and powerful charts.

SmartCharts™ is lightweight and easy to use. Complete Smartcharts documentation is at http://docs.fusioncharts.com/archive/3.7.1/

With SmartCharts there are a wide variety of ways that you can use queries and scripts to dynamically display your charts in Presto. Use the SmartChart element to easily create visually appealing SmartCharts™ that you can display in Presto.

You can also add SmartCharts via custom scripts, as described in "Adding SmartCharts with Custom JavaScript" on page 296.



As of Presto 6.6, the SmartCharts have a demo watermark 'SmartCharts Demo' on all the charts. This watermark is displayed on both the paid and trial versions of Presto.

In order to remove the watermark, you'll need to replace the demo version of SmartCharts with their commercial version. Contact Technical Support to get the SmartCharts commercial version that has no demo watermark.

Once the SmartCharts are replaced, you should no longer see the demo watermark appear on your charts.

Chart Properties

When you use SmartCharts you need to build an SQL query to generate the data to be displayed, and select the chart type that best suits your needs.

Data Source

The data source is an SQL query. You can find information in the section "Creating and Working with SQL Queries" on page 77. Here is an example of a query that returns the total of orders by customer, and also includes the customer names:

```
select  trim(MU_CUSTF.CMNAME), sum(MU_ORDHF.OHOTOT)
 from XL_PRDEMO/MU_ORDHF
 left join XL_PRDEMO/MU_CUSTF on
 MU_ORDHF.OHCUST = MU_CUSTF.CMCUST
 group by CMNAME
FOR FETCH ONLY
```

As a 3D pie chart, this looks like this:

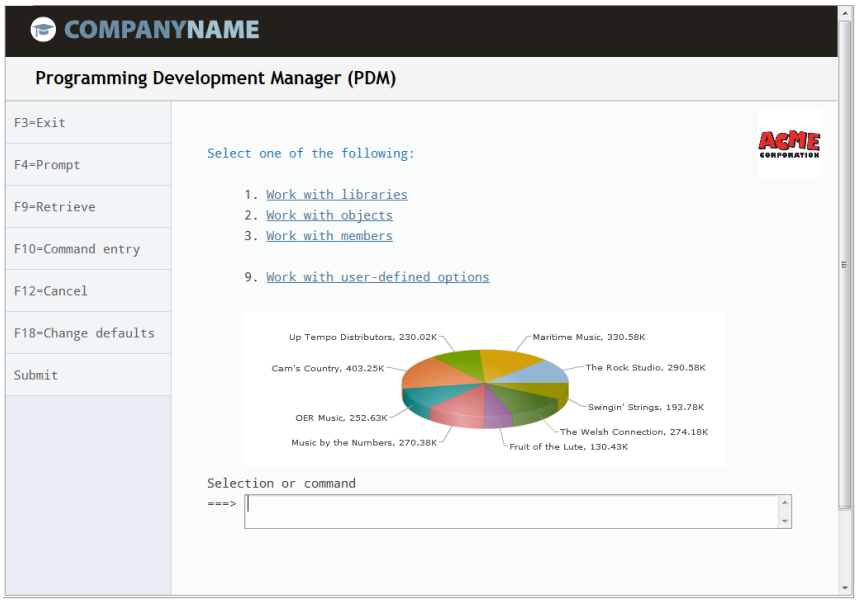


Figure 2-56. Sample chart displaying orders by customer.

You can adjust the boundaries of the chart area to resize the actual image.

Chart Type

Select the style suited to the message you are trying to convey with your data.

The tutorial application has a more detailed example of creating SmartCharts, in the section "Enhancing Your Application with SmartCharts" in the online tutorial.

Spinner

A spinner is a text input for entering numeric values, with up/down buttons and arrow key handling[1]. This element is useful in fields with a limited range of values (e.g. credit card expiration years, but not zip codes).

In this illustration we'll assume that the Discount Code has a valid range of 1-8. We are in the visual editor and have selected the field and are about to transform it to a spinner:

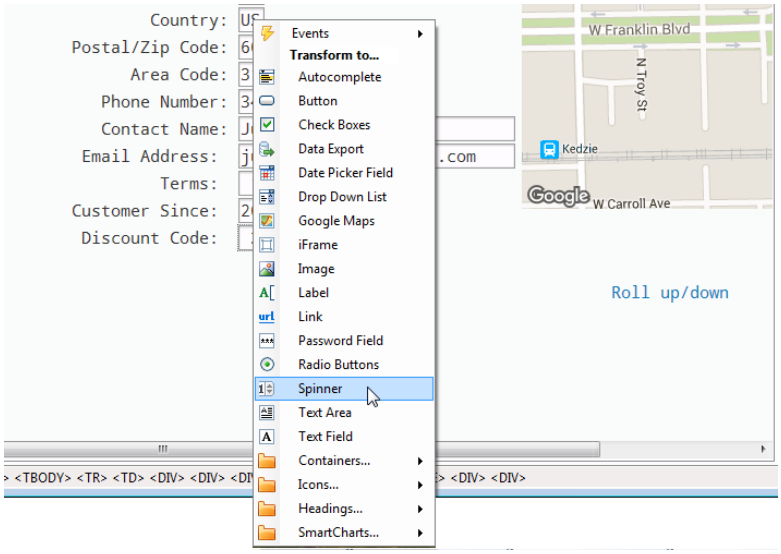


Figure 2-57. Transforming two fields to spinners using the right-click options.

After you transform the input field to a spinner you can test the default functionality:

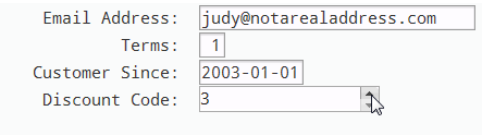


Figure 2-58. Decrementing a field value using a spinner.

In this illustration we are in the process of decrementing the field value by clicking the spinner down arrow. However, if you are testing this yourself, you'll notice that you can keep on going down past zero, or up to a zillion.

The spinner properties can help you with this.

Spinner Properties

Below is a summary of the properties. You can find the online documentation for this jquery widget at [http://api.jqueryui.com/spinner/.](http://api.jqueryui.com/spinner/)

[1]http://jqueryui.com/spinner/

min/max

Use these value to constrain a field value within a particular range. If you leave this blank, the spinner will not be limited in that direction.



As will all web applications features of this nature you need to be aware that while there is some implicit validation here, a malicious user can easily circumvent these restrictions. Where it matters, it is important to always validate on both client server sides.

numberFormat

Format of numbers passed to Globalize (https://github.com/jquery/globalize,

http://wiki.jqueryui.com/w/page/39118647/Globalize), if available. Most common are "n"

for a decimal number and "C" for a currency value.

incremental

Controls the number of steps taken when holding down a spin button. This option is documented at the jQuery UI website, at:

http://api.jqueryui.com/spinner/#option-incremental

page

The number of steps to take when paging via the pageUp/pageDown methods.

This property is further described at the jQuery UI website:

http://api.jqueryui.com/spinner/#option-page

step

The size of the step to take when spinning via buttons. This option is also documented at the jQuery UI website, at:

http://api.jqueryui.com/spinner/#option-step

Tabs

Add tabs along the top or sides of a page to reorganize complex or busy pages. This section describes the key steps. To begin, navigate to the page you want to customize and click the Visual Editor tab. Then you can get started.

If you want to follow along, we'll be using the tutorial application's Customer Maintenance program as the basis of the example:

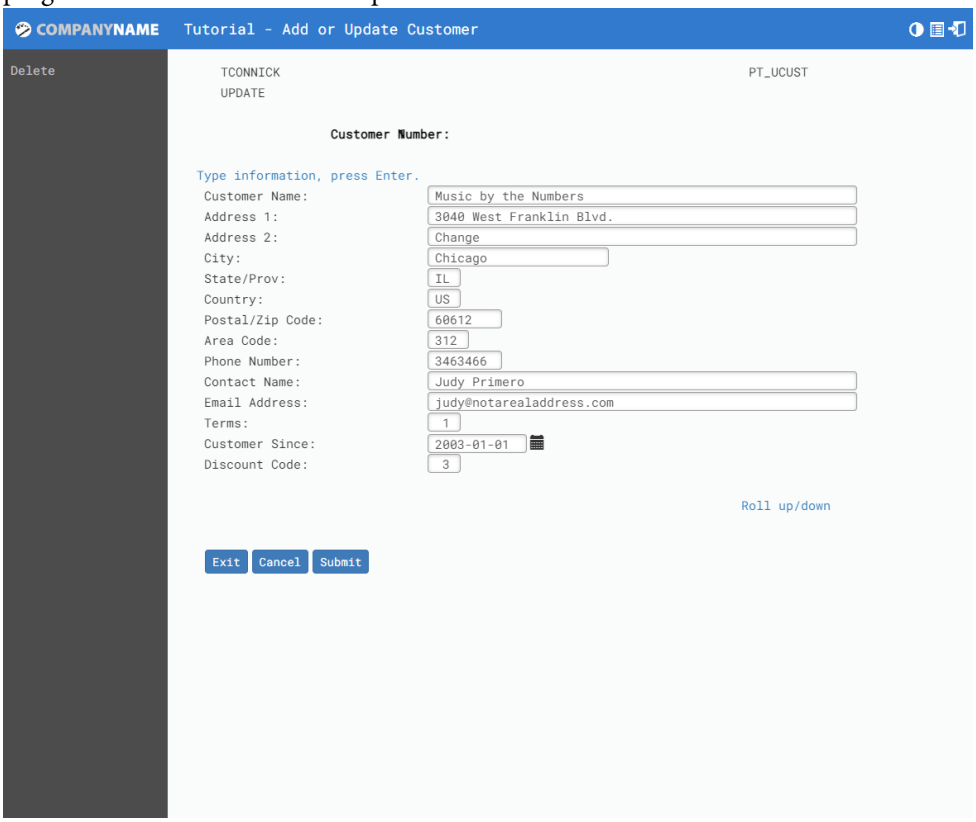


Figure 2-59. A screen that we want to break into separate sections.

While this page isn't anywhere near as congested as what you find in many real applications, it will serve to illustrate how you can break the content into separate sections.

Step 1: Add the tab handles.

In the visual editor drag three tabs over from the Containers palette:

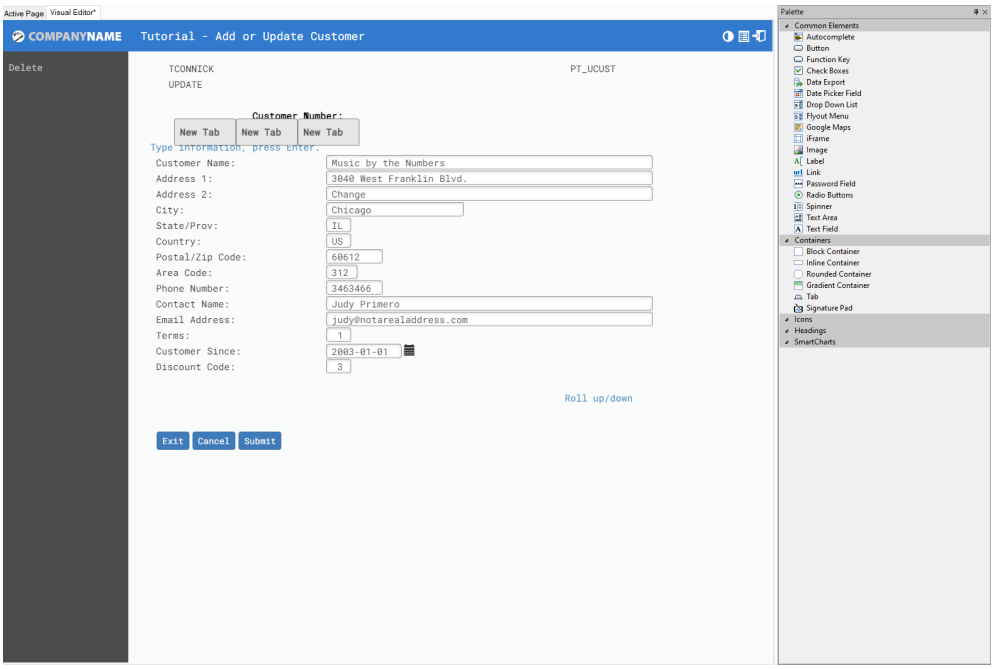


Figure 2-60. Adding tabs to the page.

If the palette isn't initially visible select that option from the View menu. You can also use the right-click menu to insert the tab elements.

In this illustration we have added three tabs. As you do this yourself, notice how they automatically snap into alignment.



Note that 'vertical' tabs are not supported by Responsive skins. The 'vertical' tabs option is disabled by default in the latest version.

Step 2: Adjust the tab text.

Double-click each tab in turn and type over the default text. In our example we are naming them:

- Address
- Contact
- Misc

Notice how Presto automatically resizes the tabs to fit the text.

Step 3: Confirm the Tab Group.

In the Field Settings notice that all three tabs belong to the same tab group:

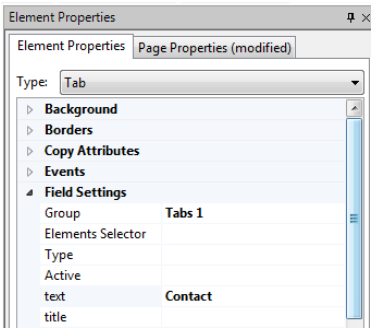


Figure 2-61. Updating the tab text and confirming the tab group.

A set of tabs must be assigned the same group.

Step 4: Set the default active tab, and the rest as inactive.

After you define your tabs pick the one that will be displayed by default and set the Active property to true:

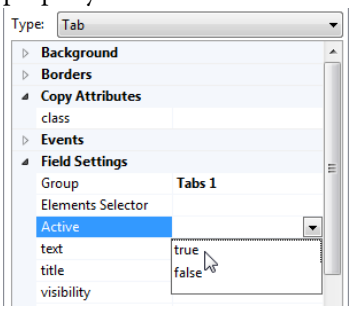


Figure 2-62. Setting the default tab to be active.

In this case we picked the Address Info tab and set the Active property to 'true'.

Then set the other tabs to Active:false.

Step 5: Select the elements to include in the first tab.

The next step is to select all the items to include in the first tab.



Use the Ctrl key to select and edit all these items at the same time, or you can drag a box around them all to select them at once.

In this illustration we are in the process of selecting the first 7 fields on the panel with their text labels, by drawing a box around them all:

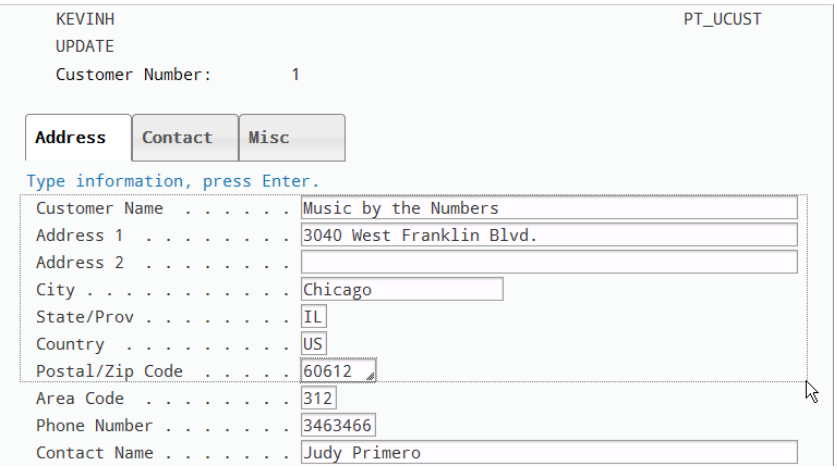


Figure 2-63. Drawing a box around a group of fields to select them.

When we release the cursor at the bottom right, all the elements in the area will be selected.

Step 6: Assign the elements to the appropriate tab.

The next step is to assign a class to these fields. The easiest way to do this is using the right-click option:

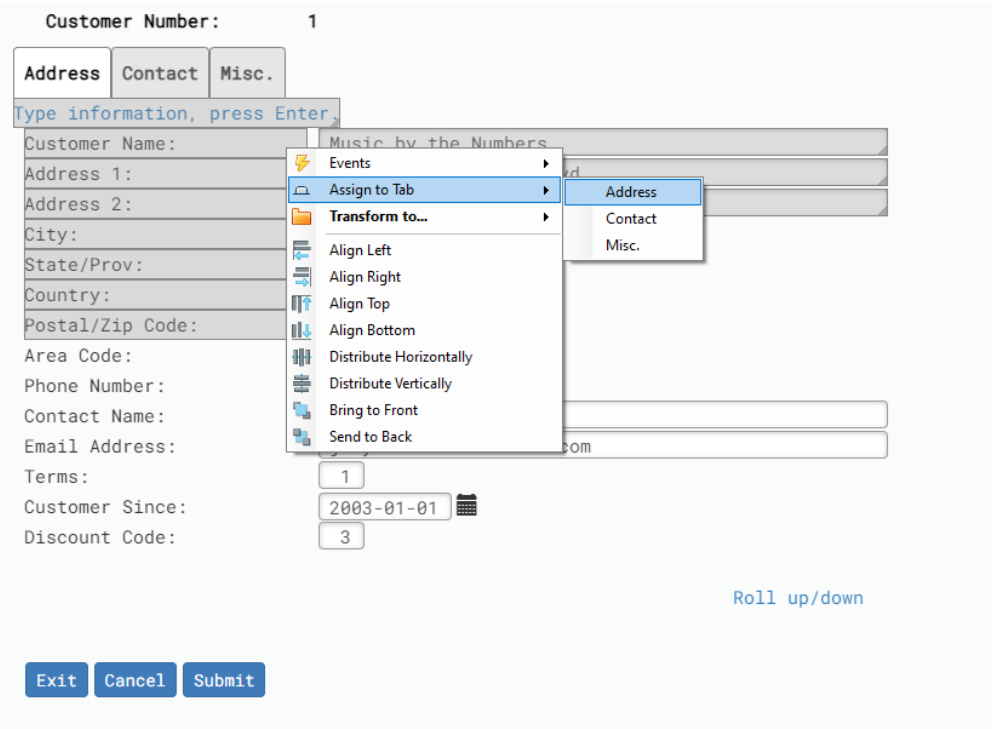


Figure 2-64. Assigning a unique class to the items together in one tab.

Right-click and select the tab from the 'Assign to Tab' menu option.

Step 7: Repeat Step Step 5 and Step Step 6 for each tab.

After selecting the fields for the first tab, repeat this process for your other tabs.

Add the phone number, contact name and email to the Contacts tab, and the rest to Misc.

Save your changes.

Step 8: Adjust the layout of your tab contents.

At this point you should be able to click on each of the tabs and the Presto visual editor should only display the fields associated with each tab. For example, clicking the Contact tab should produce this outcome:

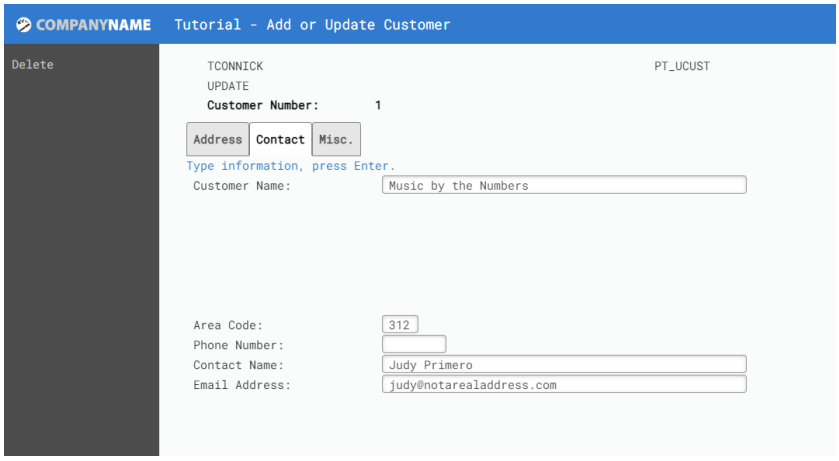


Figure 2-65. Reviewing the outcome from the creation of the new tabs.

If this works it means you're ready to start reorganizing the fields and other elements in the tab and making any further customizations, towards something like this, perhaps:

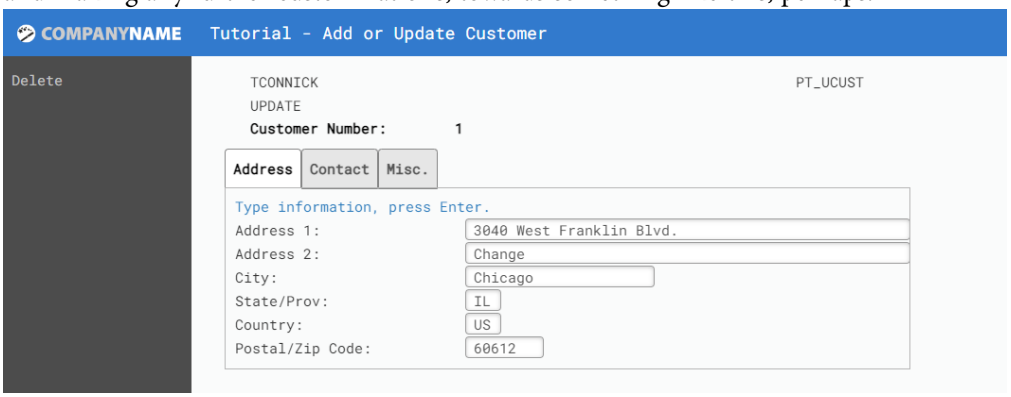


Figure 2-66. Tabs completed.

In this illustration we have used containers to draw lines around the tab area.

This section has provided an overview of the most basic tab implementation. There is much more that you can do! If you have any other questions or needs when designing your tabs call Technical Support for help.

Table

You can use the table element properties when you select a table that either you or Presto have added to a panel. In this illustration we have changed some of the properties of a table over the list of items in the tutorial application:

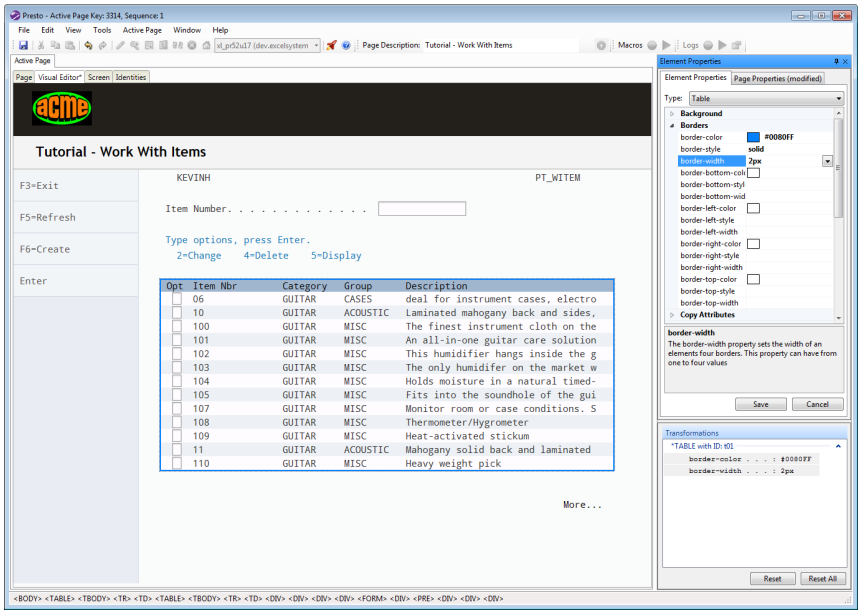


Figure 2-67. Using the visual editor to work with table properties.

Presto automatically recognizes tables when you select them, and displays the properties in the editor.

Text Area

Text areas are useful for editing long fields that span several lines.

For a detailed example illustrating this element, as well as demonstrating how to switch a text field to a text area, please review the sections “Setting the Element Type” on page 166 and “Resizing Elements” on page 169.

Note that regardless of the text area size in the browser, the field length in the IBM i program controls the number of characters that the program will accept.

Submit on Enter

Text areas do have one unique visual editor option: the choice on whether to submit the content on Enter:

- **true.** This setting would be intuitive for multi-line command lines, where the user will expect that the Enter key will submit a command.
- **false (default).** This setting will likely be more intuitive for text areas that represent long text fields. In these cases, the user may expect Enter to go to a new line.

As noted above, the default behavior is to break to the next line.

Text Field

Text fields are used to input user information[1]. Presto automatically assigns the text field type to any input-capable fields on your screens. As you start customizing your screens further you'll see that you can create elements of other kinds (radio button, checkbox, etc.).

[1] http://www.w3schools.com/TAGS/tag_input.asp

Chapter 12 - Global Transformations

Global Transformations are designed to apply system-wide transformations based on the text of a given screen. Text that reoccurs throughout your green-screen application can be found, replaced with substitution text, and applied a transformation.

Presto ships with a number of predefined global transformations that can be enabled in the Global Transformations form located under the Tools menu.

Global Transformation Manager

Accessing the Global Transformation Manager

To view the Global Transformations, select Global Transformations from the Tools menu.

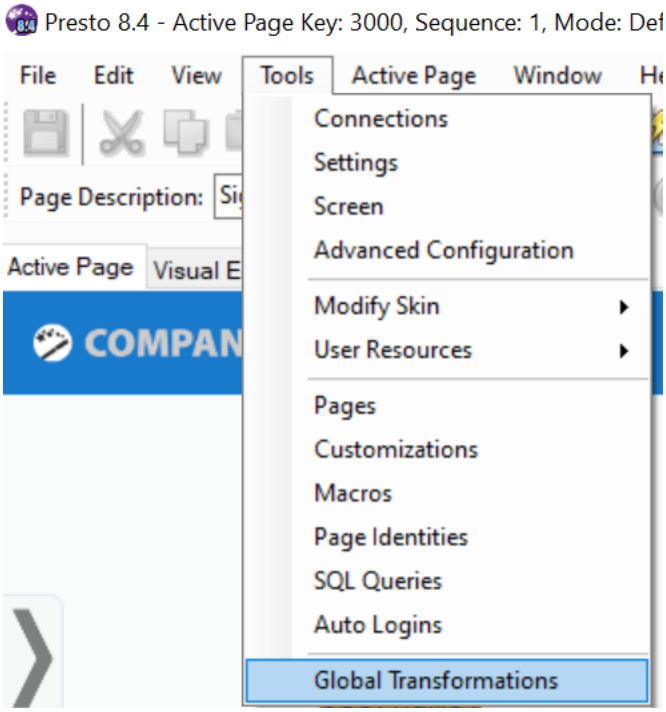


Figure 2-1. Viewing Global Transformations.

Managing Global Transformation Entries

The Global transformations manager lists all existing Global Transformations. You can re-order the execution order and create, copy, or delete any entry using the left list view. Entries can be re-ordered using either the arrow buttons to the right of the list, or by dragging and dropping elements within the list.

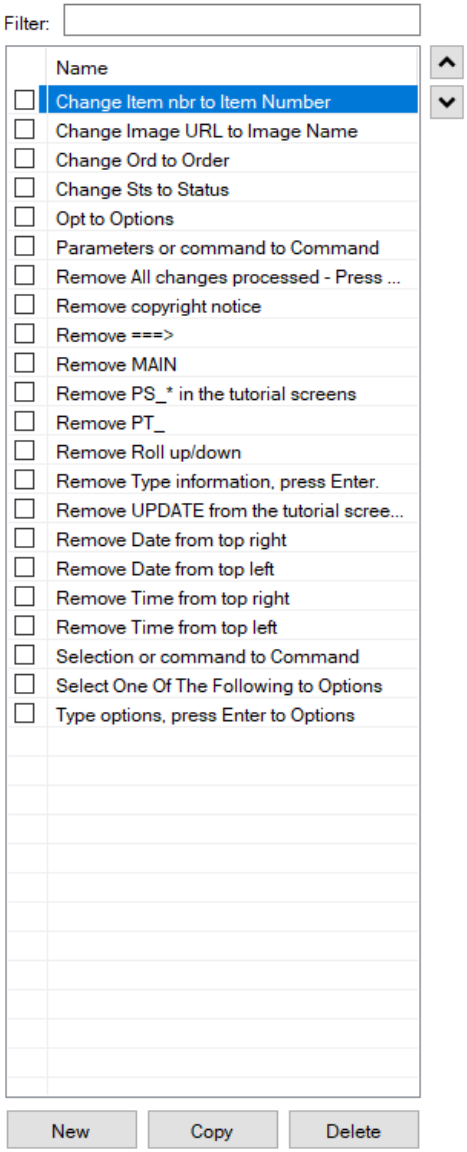


Figure 2-2. Viewing Global Transformations entries.

Search Strings and Transformations

The Global Transformation manager allows you to enter the selection criteria and associated transformation to perform. You can see these options on the right side of the Global Transformation manager. This area of the utility is broken up into several sections:

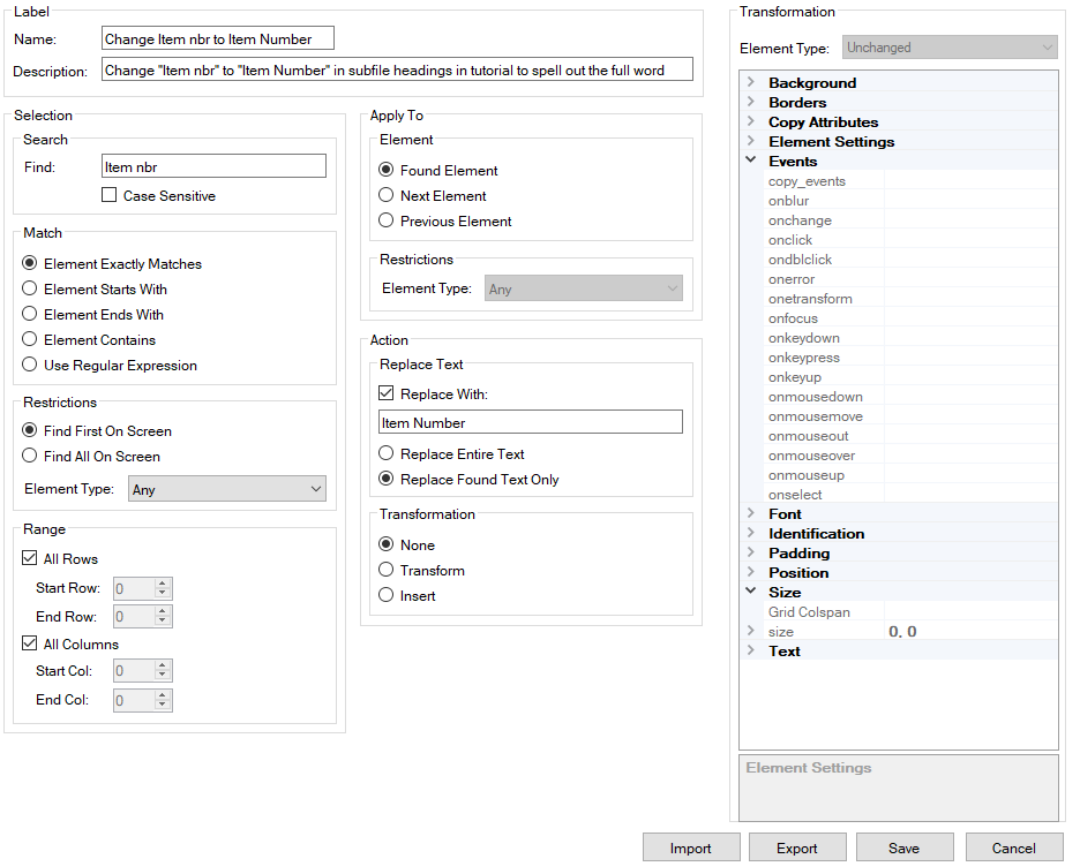


Figure 2-3. Search strings and options.

Selection

The Selection group is where you define the search criteria to apply this transformation:

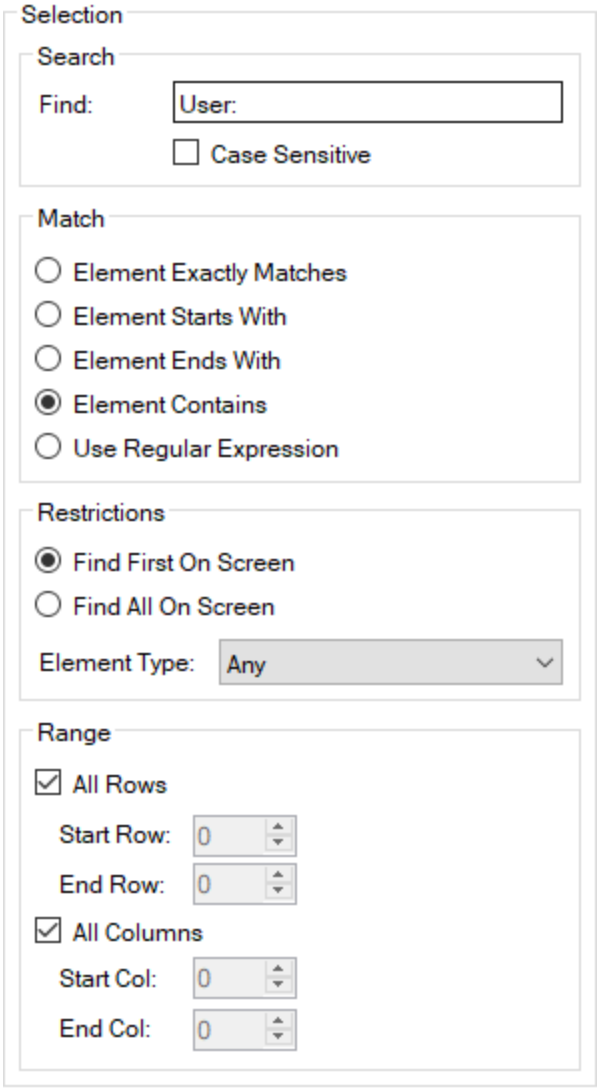


Figure 2-4. Viewing the selection and search criteria..

Use Selection options to enter search criteria for your transformation:

- **Find.** This is the text to search for on the screen to apply a transformation in relation to.
- **Case Sensitive.** When selected, only match text that has the same pattern of upper and lower case characters as the search-text string.
- **Match.** Set the type of text that matches with the search string
  - **Element Exactly Matches.** Select only screen text that exactly matches the characters specified in the search text.

- **Element Starts With.** Select only screen text that starts with the same characters specified in the search text.
- **Element Ends With.** Select only screen text that ends with the same characters in the search text.
- **Element Contains.** Select screen text that contains the letters you specify anywhere within the search text.
- **Use Regular Expression.** Select screen text that matches the pattern defined by the regular expression.



If multiple Global Transformations match the same element, they are applied based on their execution order in the list of Global Transformation Entries.

Find Restrictions

- **Find First On Screen.** Apply the transformation only once per screen. Stop searching when a match is found.
- **Find All On Screen.** Apply the transformation once per element. Sequentially search every element in the screen and apply the transformation to each matching element once.
- **Element Type.** Apply the transformation only to the element type selected in the menu. Select the element type as All, Text, Input, or Button.

Range

Use the Range options to restrict the search to a specific area of the screen. Note that any value left at '0' will be considered the farthest bound in the given direction, so a "Start Row" of 0 will be 1, while an "End Row" of 0 will be either 24, or 27, depending on the screen.

- **All rows.** Search all of the rows on the screen.
- **Start row/End row.** Enter a range of the starting and ending screen rows to search.
- **All columns.** Search all of the columns on the screen.
- **Start column/End column.** Enter a range of the starting and ending screen columns to search.

Apply To

The Apply-To group allows you to specify which element to apply a transformation to.

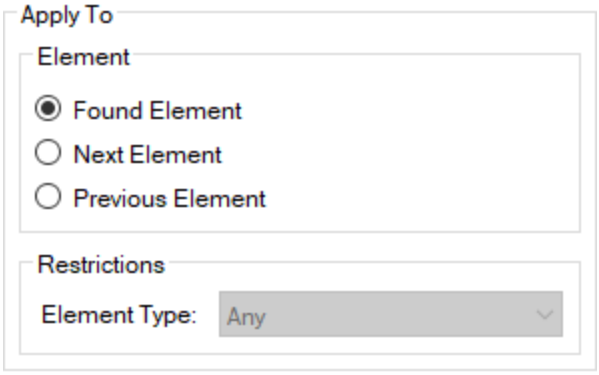


Figure 2-5. Viewing the Apply To Group

- **Element.** Specify which element to apply a transformation to, relative to the element where the Search Text was found.
    - **Found Element.** Apply the transformation to the element that contained the Search Text.
    - **Next Element.** Apply the transformation to the element that is directly after the element that contained the Search Text.
    - **Previous Element.** Apply the transformation to the element that is directly before the element that contained the Search Text.
- **Restrictions.** Apply the transformation only to the element type selected in the menu. This option is enabled only if Next Element or Previous Element is selected in the section above.

Actions

The Action group allows you to specify what actions to take when the Search text is found on a given screen.

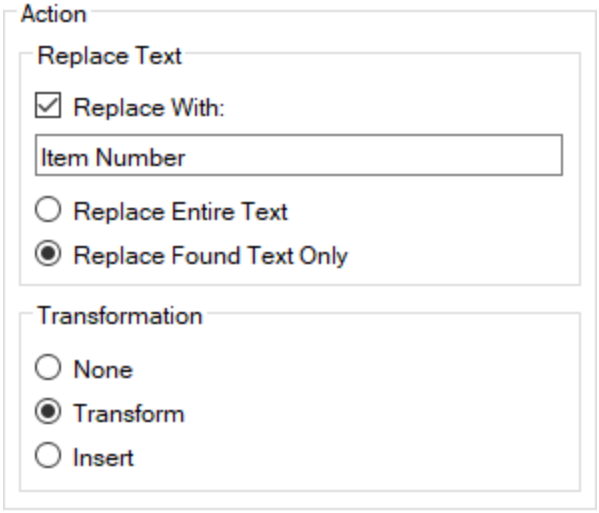


Figure 2-6. Viewing the actions.

- **Replace With.** When checked, replace the element string with the specified string using one of the two options below:
    - **Replace Entire Text.** If selected, replace the entire text of the element that the search string was found in with the specified string.
    - **Replace Found Text Only.** If selected, only the text that was found via Search Text will be replaced with the replacement value.
- **Transformation.** This section determines whether to apply a transformation or insert a new element.
    - **None.** Apply no transformation.
    - **Transform.** Apply the transformation to the element that contained the Search Text.
    - **Insert.** Insert a new element on the screen. Note that the insert option positions the element to the top left corner of the page by default. To change the position of the element, you will need to manually adjust the Visible Column and Visible Row propertiesin the Element Properties grid described below.

Transformation

The transformations group allows you to set what properties to apply to the element selected in the Action group.

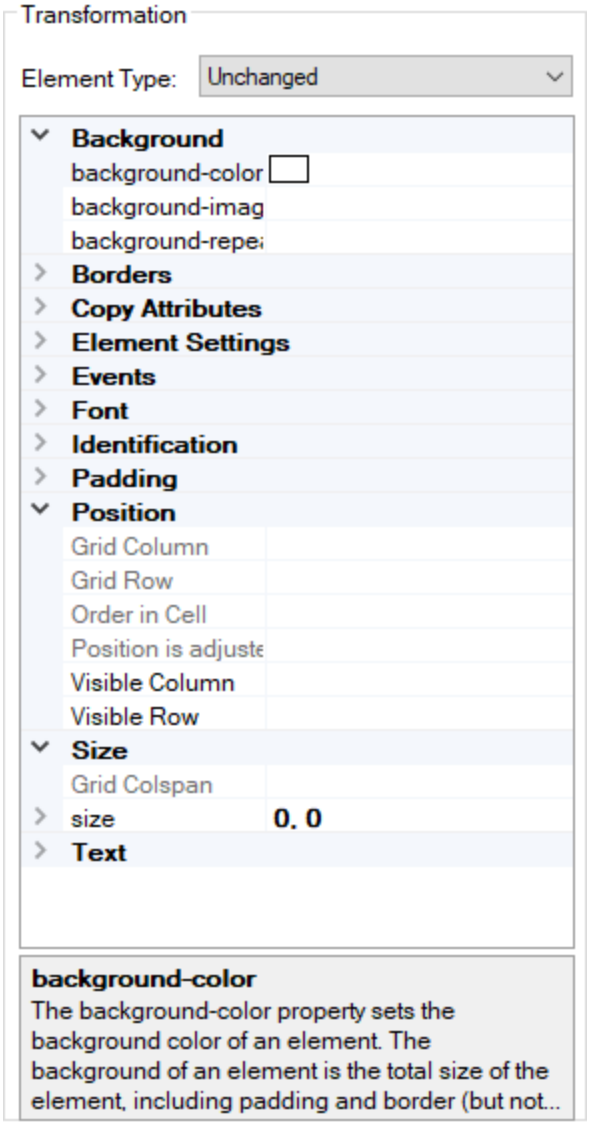


Figure 2-7. Viewing the transformations.

- **Element Type.** This dropdown allows you to transform the element to another element type, or leave it as "Unchanged" to only apply properties to the found element.
- **Element Property Grid.** The property grid is where you can specify individual properties of the transformation. The available properties will vary based on the selected Element type.
- **Save.** Save all pending global transformation changes to the server.
- **Cancel.** Cancel all pending changes, and revert to the previous saved global transformation list.

Adding a new Global Transformation from the Visual Editor

You can add a new Global Transformation from any transformation created within the Visual Editor. This allows you to easily setup the properties of a transformation using the Visual Editor before migrating it to a Global Transformation.

To add a Global Transformation, right click on any element with a compatible transformation and this context menu will appear:

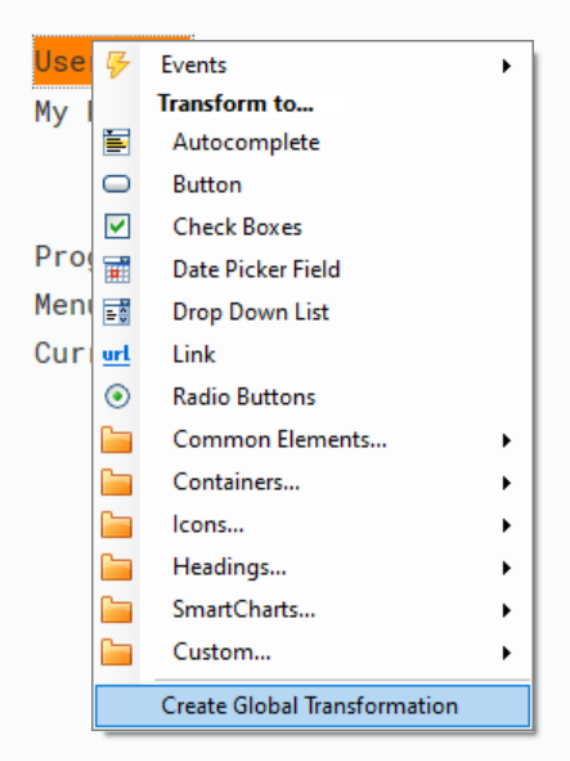


Figure 2-8. Creating Global Transformations From the Visual Editor.

Click the new "Create Global Transformation" menu option at the bottom of the context menu. This will create a new Global Transformation that is pre-populated with the search text and transformation properties that were applied to the transformed element.

Migrating Global Transformations

If you have separate development and production environments, you may encounter the need to migrate your Global Transformations between environments. All global transformations are stored on the IFS under Presto's installation folder, in the "global_transformations" subfolder. This folder will contain a "global.json" file, along with backups created during the previous 5 save operations, denoted by an underscore and number (1 being the most recent backup, and 5 being the oldest).

To migrate your global transformations to a new system, you only need to copy the "global.json" file from the development system to the production system.

Exporting and Importing Global Transformations

Exporting Pages

To export a global transformation or set of global transformations, start by selecting the Tools > Global Transformations menu option in the Presto Designer.

Here is an illustration from an environment with several global transformations:

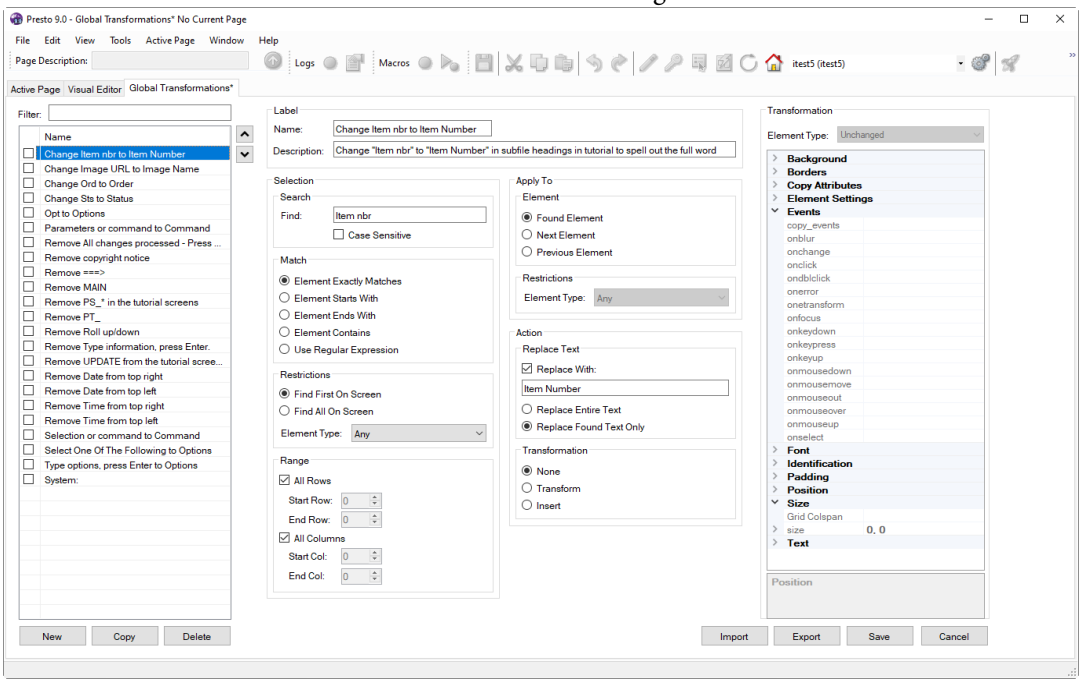


Figure 2-9. Preparing to export global transformation.

In this illustration, we have selected all the global transformations in this environment and are preparing to export:

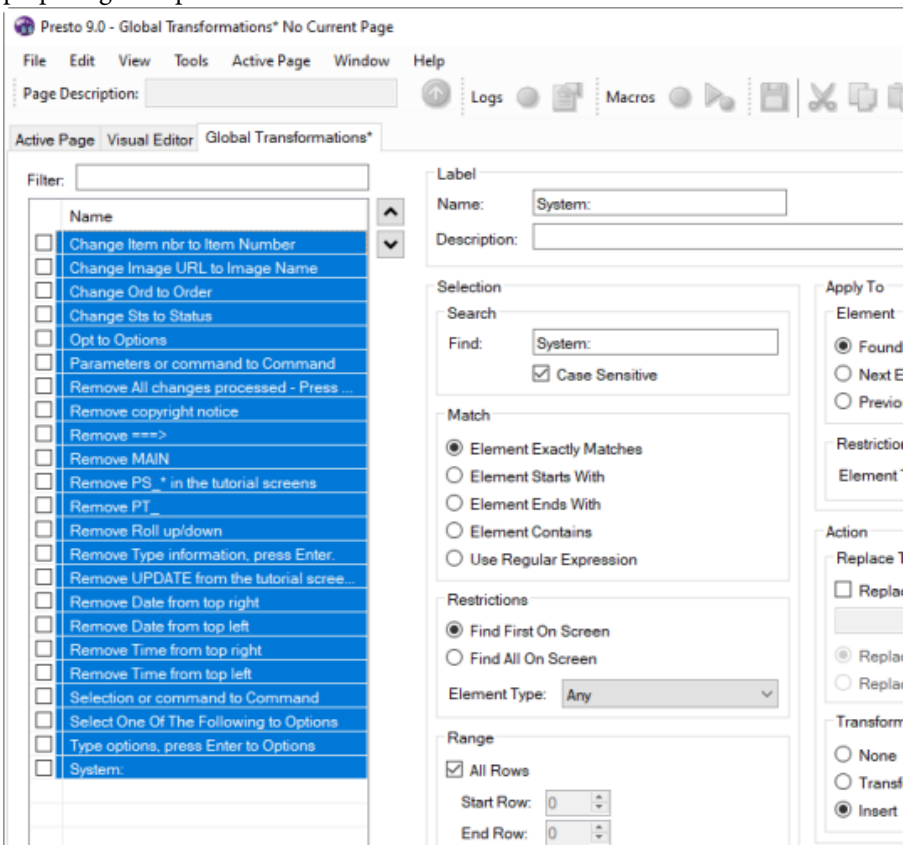


Figure 2-10. Selecting the Global Transformations to export.

You can hold down the Ctrl key while you click multiple items, or use Ctrl+A to select all your transformations.

Click the Export button. Choose a location where you would like to save the JSON file containing all your Global Transformations:

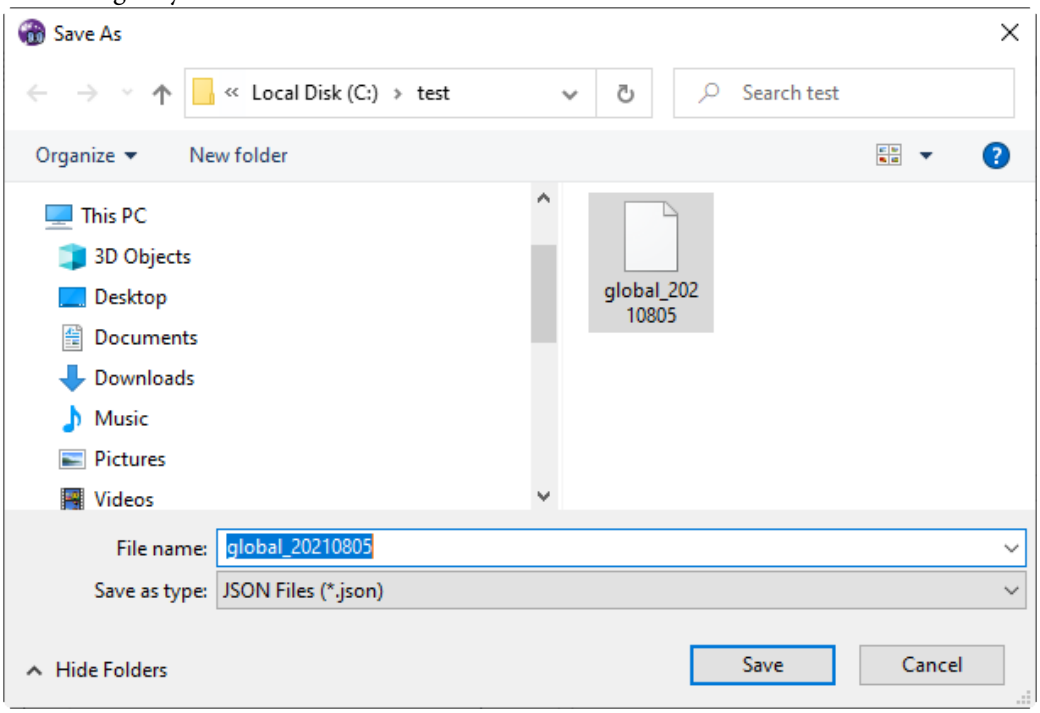


Figure 2-11. Saving the JSON file with the global transformations.

The default file name is 'global_YYYYMMDD.json'. You can override the file name (for example, if you export multiple sets of global transformations on the same day).

Save the file.

Importing Pages

the Import button, Presto presents a window confirming that the global transformations will overwrite existing ones:

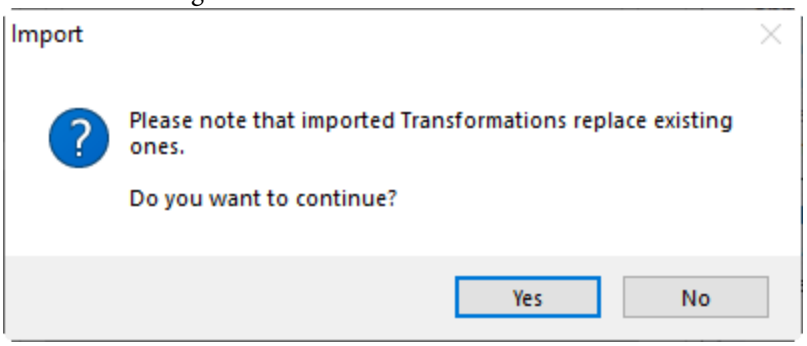


Figure 2-12. Import Confirmation.

When you click Yes above, Presto prompts you to import a JSON file:

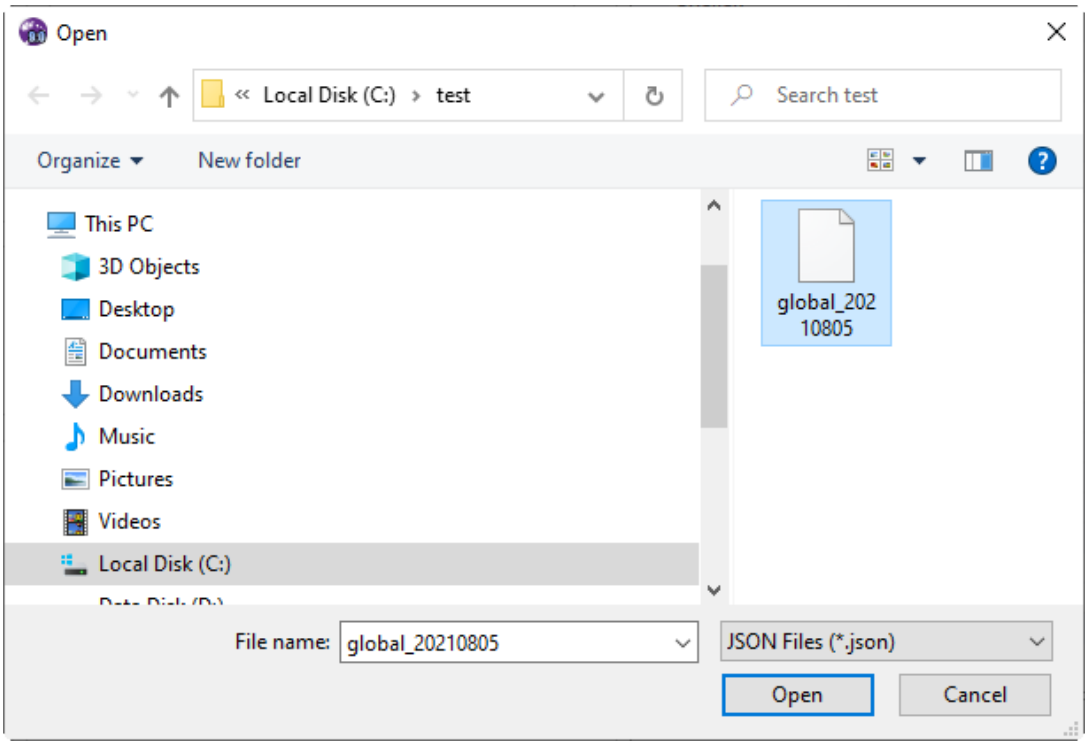


Figure 2-13. Preparing to import a set of global transformations.

When you select the file, Presto will execute the import, and then display a confirmation message.

A "Succesfully imported" message confirms that all global transformations were copied into the environment. The newly imported transformations will be highlighted. If the imported global transformation is enabled, you should see it applied.



Note that imported global transformations may have other dependencies at the environment level (SQL queries, for example).

Considerations

Note the following about the Global Transformations:

- Global Transformations will not apply to any screen that has active Custom HTML applied to it.
- A large number of Global Transformations may have a negative effect on screen-loading time. This is due to each screenload needing to process every global-transformation rule to determine if it applies to the new screen.

- Changes to Global Transformations will only be shown after the next refresh of any active session. This includes when you modify global transformations in the Presto Designer.
- Global Transformations are applied before screen-level transformations. This allows you to overwrite a global transformation with a screen-level transformation in the event that a global transformation has transformed a screen in an unintended way.
- Some properties are not automatically migrated from a screen-level transformation to a global transformation. This includes the visual row/column properties, as well as the element type and value.

One Transform Event

Presto includes an event "onetransform". This event extends global transformation functionality by adding the ability to execute custom JavaScript following a global transformation. The event can be set in the transformation's element property grid:

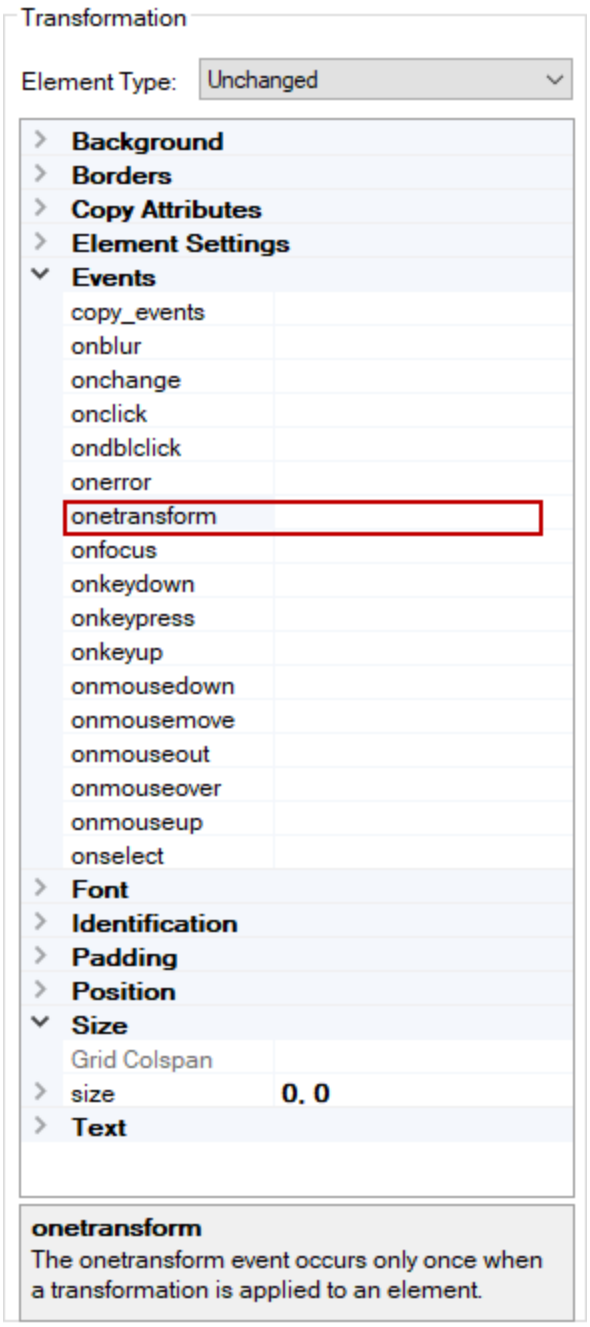


Figure 2-14. The OneTransform Event

CHAPTER 13

Chapter 13 - Enhancing Your Pages with JavaScript

In previous chapters you have learned to modernize your green screen applications using the Page Properties and the Visual Editor.

The next technique you should become familiar with is adding scripts to individual pages, and to your site as a whole.

Reference the section "Applying JavaScript to a Single Page" in the online tutorial for instructions on how to add Custom JavaScript via the Presto designer. The section "Applying a Transformation to Every Page" discusses how to add scripts that affect every page on your site.

This chapter includes some additional technical details related to implementing custom JavaScript in your site, but please use the tutorial chapter as your primary reference.

Adding Simple JavaScript to Presto Pages

A complete JavaScript tutorial is beyond the scope of this guide (and there are many excellent tutorials and other resources available online). However, the following sections provide a basic tutorial describing how to incorporate JavaScript into your Presto pages.

Where To Create Your Customizations

You can create custom scripts via the Custom JavaScript interface, or append them to your HTML in Custom HTML. As a general rule, you should use the Custom JavaScript files (rather than Custom HTML) unless you are also making some additional structural changes to your page's HTML that can't be handled via the visual editor or you custom script.

When you click the button to create a custom JavaScript Presto displays a new tab with some comments for documentation. If you use this interface to add a script, you do NOT have to include these tags.

```
<script type="text/javascript">

</script>
```

If you add scripts in your custom HTML, you do need to enclose them in these tags.

Where to Create Custom Scripts That You Will Reuse

When you installed Presto you specified an environment library (by default XL_PRESTO). An IFS folder with the same name is created within your root/esdi/ folder which contains some shared objects for your Presto environment.

In your IFS, at /esdi/environment_name/userresources/ you can add your changes to the custom.js file you'll see there. You can access this file through the Presto Designer, at Tools>User Resources>custom.js. See Chapter 2 for further information on using custom scripts to customize your screens.

About the jQ Object Used in Presto

In some of the example code below you will see references to a JavaScript object called jQ. This object is an alias for the standard jQuery object that is usually named $. We renamed it to avoid problems with the $ character when using Presto in languages other than US English, and to have a less arcane name than just a $ sign. This object is used to select HTML elements and then act upon them. We'll explain each usage of the object in the examples below.

Presto JavaScript Snippets

This feature allows you to store and apply common JavaScript code snippets.

JavaScript Snippets Files

To save your JavaScript to an external file and re-use it in various pages across your site, simply place your JavaScript in a text file and save it to your PC in Presto's \snippets\js directory with a .js extension:

- Shipped snippets are located typically on your PC at C:\Program Files (x86)\ESDI\Presto\snippets\js\

- User-created snippets are created in C:\Users\[User]\AppData\Roaming\ESDI\Presto\Snippets\
- Presto only loads .js files into the snippet view.

How to Use JavaScript Snippets

- In the Presto IDE, when creating or editing custom JavaScript, there's a new view, Code Snippets, which lists all the snippets files.
- To insert a snippet:
  - Place the cursor on the line in your custom JavaScript where the snippet should be inserted and then double-click on the snippet.
  - Alternatively, you can drag and drop the snippet into your custom JavaScript.
- Undo (Ctrl+Z) can be used for removing a snippet.

Here is an example of the snippets view:

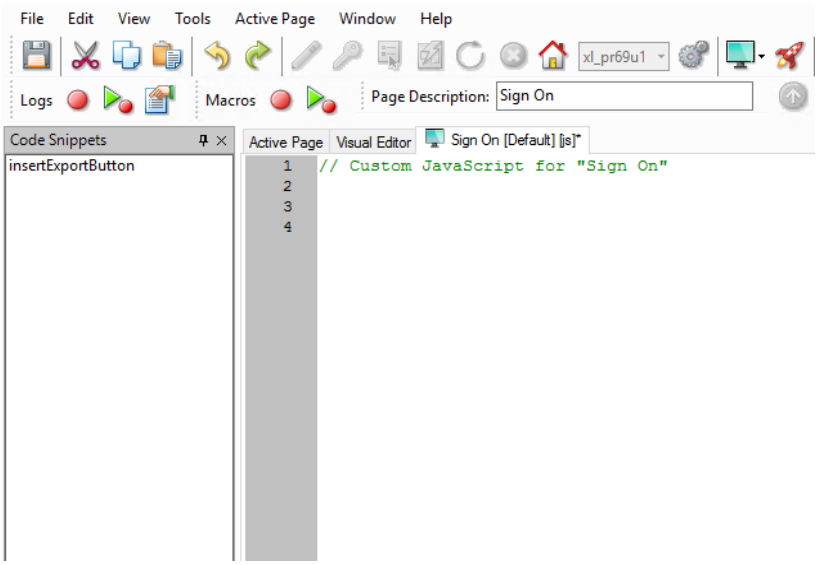


Figure 2-1. Adding a JavaScript snippet to the custom JavaScript.

You can also create new snippets by right-clicking anywhere on the page and selecting the 'New Snippet' option:

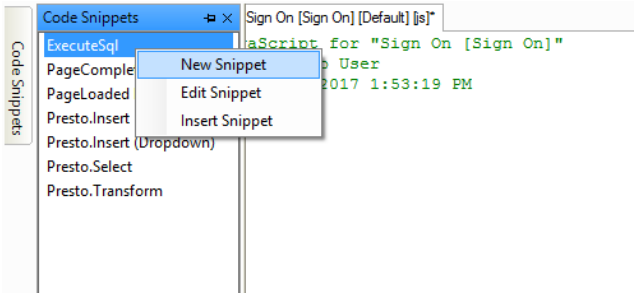


Figure 2-2. Creating a new snippet.

Here is the full snippet in the Snippet Editor:

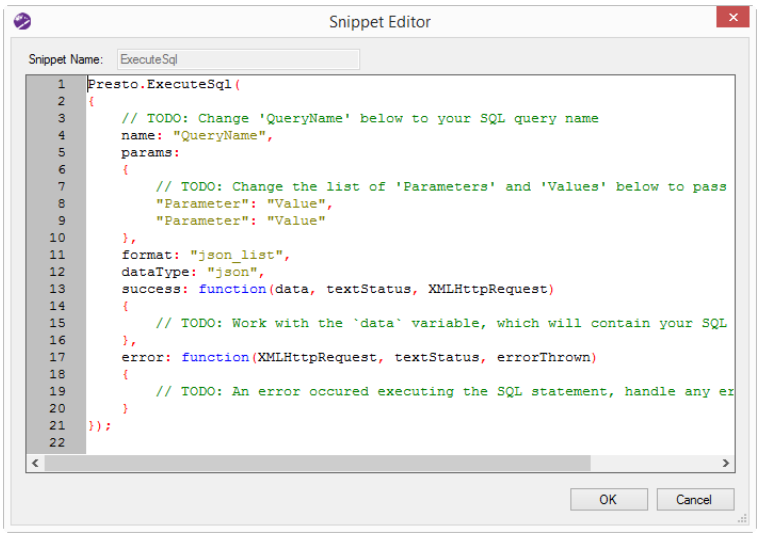


Figure 2-3. The Snippet Editor.

Hello World

Here is a simple, standalone JavaScript function:

```
<script type="text/javascript">
   alert("Hello World");
</script>
```

If you create custom HTML for your page you need to include the <script> and </script> tags, but if you use Presto's Custom JavaScript option you only have to add the middle line.

Here we are adding this script to a custom JavaScript over the tutorial main menu (after removing some detritus left over from the tutorial):

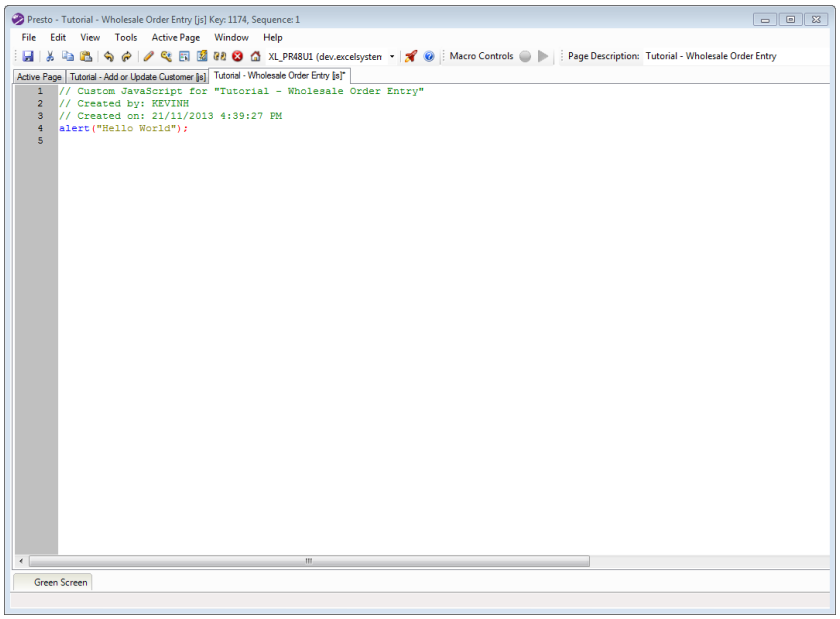


Figure 2-4. Adding a simple JavaScript to the HTML for a flat panel display.

This is all that is required.



When you save or refresh here, you'll see the message displayed above, because it's set to display it *before* the page loads.

When we save this change and run our program in the browser, it does this:

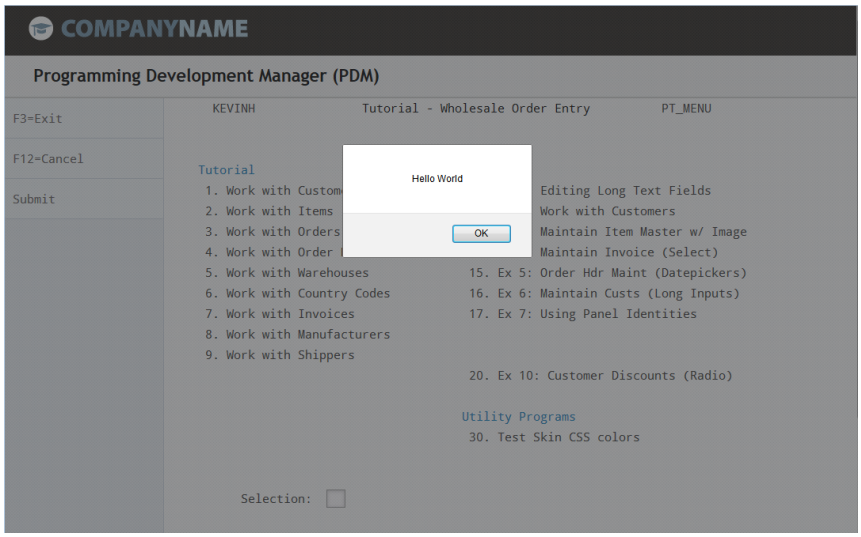


Figure 2-5. Displaying a JavaScript alert message on your page.

All we've done here is display an unconditional message at a particular point while the page is loading. There are obviously much more practical uses for this feature - in fact, in your Presto pages you can do anything that's possible in JavaScript!



We often use jQuery, a popular and exciting open source JavaScript library, to add cool features. Learn more about jQuery at www.jquery.com, or check out the KB articles, examples and videos at the Fresche Technical Support site.

When Does JavaScript Get Executed?

Before we get too far, let's clarify a minor technical matter: for most web pages, JavaScript is run when the page is loaded, to provide extra functionality. However, with Presto, the 'page' is actually loaded only once when you first start your session, and each screen is placed dynamically as you run commands and click buttons. Presto runs your script after the HTML is loaded, before the Visual Editor transformations, and before the 'PageComplete' event has been triggered.

Page HTML Visible via Dockable Window

A page's HTML is visible via a dockable window when you are creating or editing a custom JavaScript file for the page. In this illustration we are about to start adding some JavaScript to a page, and have clicked the new HTML tab indicated at the bottom here to show the page's HTML:

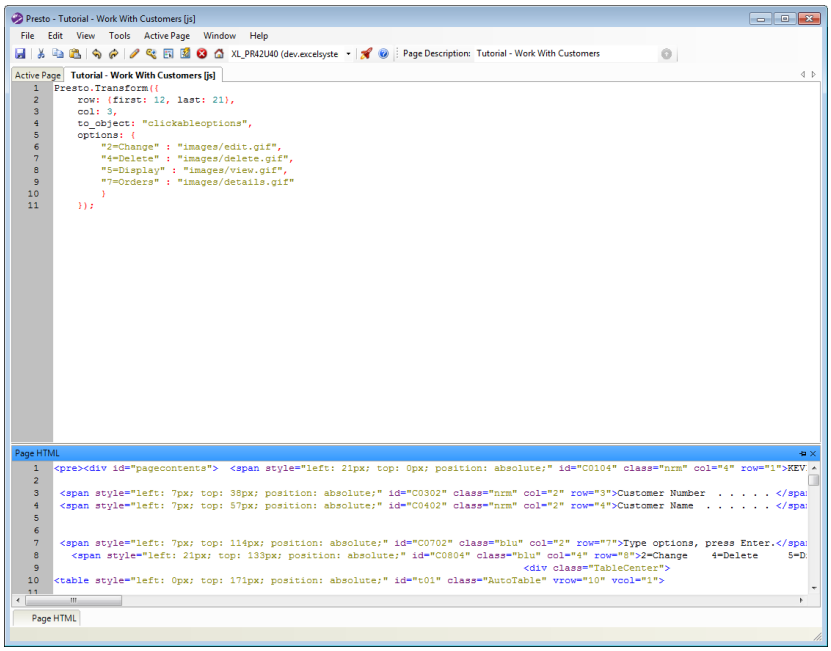


Figure 2-6. Revealing the page's HTML when adding JavaScript customizations.

You can also pin the HTML in the designer, or tear it away to see it in a separate window.

Example: Adding a New Command Key

Presto makes it easy for you to add a new command button to your pages, using a format similar to that of the automatically generated function key buttons. In this section we'll explain the process, using this feature as a basis for pointing out some key concepts.

This is our i5 Main Menu when we first log on, with no modifications (other than a custom skin):

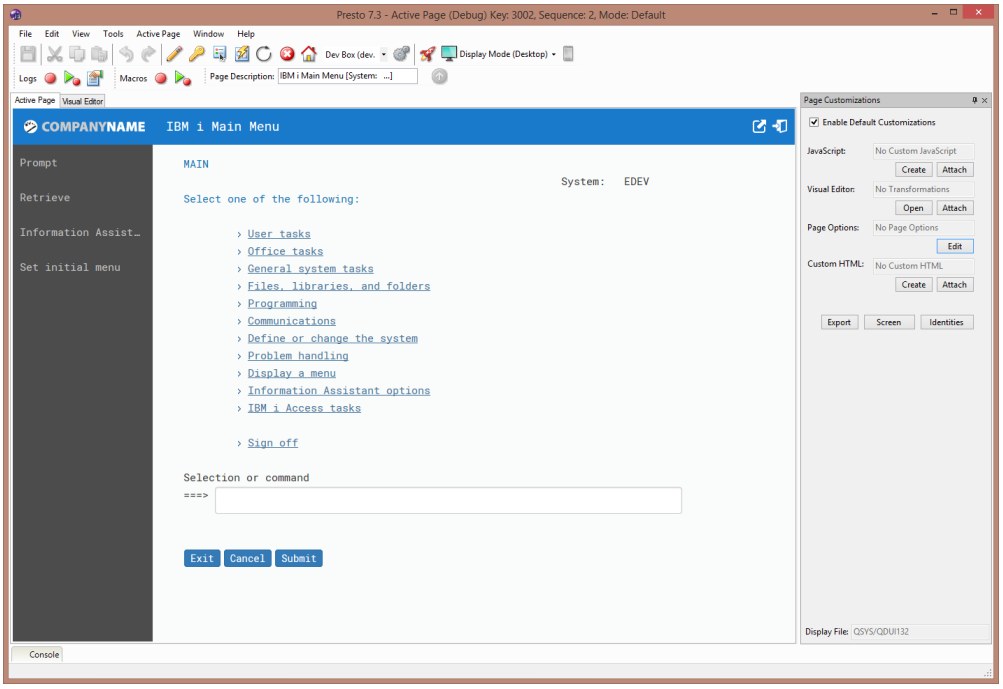


Figure 2-7. The main menu, as presented by Presto with no changes.

Let's add a new command button that will launch our tutorial programs.

Step 1: Create the Custom JavaScript.

To add a new JavaScript page begin by clicking the Create option near the bottom of the right side menu (as shown in the illustration above).

When you do this, Presto displays a new tab where you can add your code.

About the Generated Command Keys

When Presto builds the HTML for your pages, the code for your command keys looks something like this (using F13 as an example, where the function keys are on the left):

```
<a class="button blu " id="F-22-51" style=" width: 227px; " col="51" row="22"
funckey="13">F13=Information Assistant</a>
```

Notice the following items:

- The input includes row and column parameters. This information is also recorded in the id: `"F-22-51"`.
- The anchor tag is of class `"button blu"`, which is tied to a CSS tag in your skin controlling the appearance of the buttons on the screen.

- The action associated with this button is governed by the funckey parameter, which submits a `SendRequest('13');`

The end result is a button that when clicked works like your keyboard F13.



SendRequest is a JavaScript function defined in presto.js, a JavaScript library used in all your Presto programs. We'll discuss this in more detail in later chapters, but for now it's enough that you know that you can add code to use it just like Presto does when generating the page's HTML.

Step 2: Add a New Button.

With a basic understanding of the components that comprise a standard button, you can add one of your own. Here is the JavaScript to add a new button:

```
Presto.Insert({
    object: "input",
    to_object: "button",
    value: "Call PT_MENU",
    onclick: "Presto.Transform({row:20, col:7, value:'CALL PT_MENU'});SendRequest
('EN');"
}).appendTo('#fnckeys_wrapper');
```

You can copy & paste the sample above.

Step 3: Save your change and return to the Active Page tab.

After adding the script, save your changes (using the button in the top left of the window, or by right-clicking the tab). Then go back to the Active Page tab and click the Refresh button.

You should see the button at the bottom of your screen:

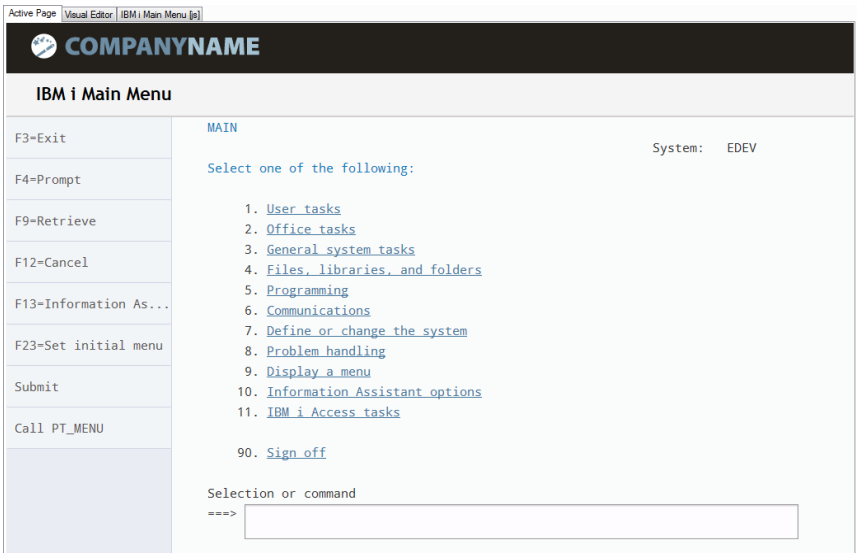


Figure 2-8. Using JavaScript to add a new button to the screen.

Now try clicking your button. It should call the menu (you'll need to have `XL_PRDEMO` in your library list).

Adding a New Drop Down List of Commands

We've already seen an example of adding a new command key to a program. Let's expand upon that technique to add a new drop down list where the options execute IBM i commands to run the various tutorial programs.



The easiest way to add drop down boxes in general is via the visual editor. See the section "Drop Down List" on page 226 of Chapter 2 for details.

Step 1: Add new text to the screen.

As our first step we'll use the visual editor to add a new menu option.

In the illustration below we have used the visual editor right-click menu option to insert a new Inline Container. Then we added the field value (the text to display):

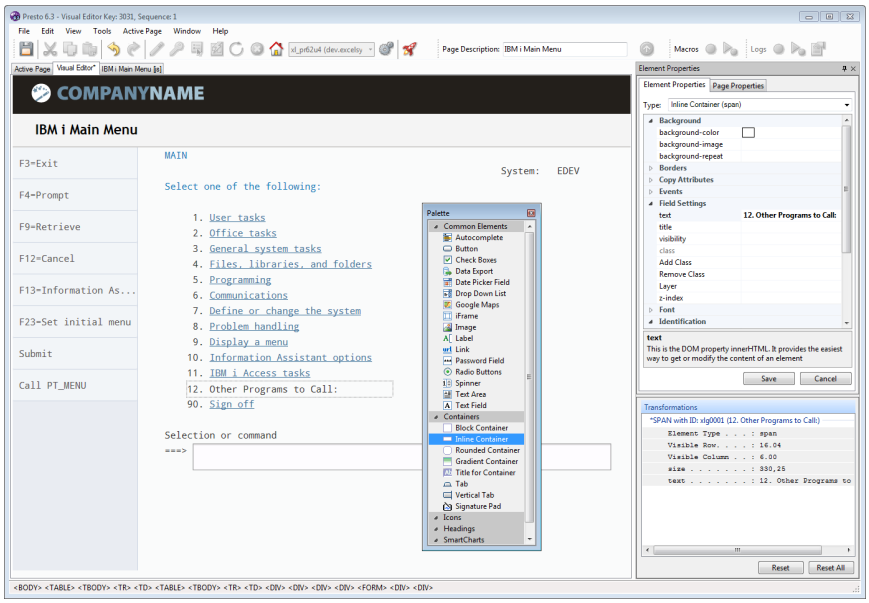


Figure 2-9. Inserting a new menu option.

The text we inserted is "`12. Other Programs to Call:`". You can use your arrow keys to fine-tune the location of the new text string, and the width & height properties to adjust the size.

Save your changes when you're done.

Step 2: Go back to the Page Customizations menu and edit your JavaScript.

Click the 'Page' tab to return to the default view. From here you can click the 'Edit' button to once more work with the JavaScript attached to this screen.

Step 3: Insert a New Object After the Text string.

Use this script to insert a new drop down list:

```
Presto.Insert({
        row : "16",
        col : "35",
        object: "select",
        options:
[
{"value" : "", "text" : "Choose a Demo Program"},
{"value" : "CALL PT_MENU", "text" : "PT_MENU"},
{"value" : "CALL PT_WCUSTS", "text" : "PT_WCUSTS"},
{"value" : "CALL PT_WITEM", "text" : "PT_WITEM"}
        ],
        onchange: "Presto.Select({              row: 20, col: 7}).val(this.value);SendRequest('EN');"
});
```

This uses the built-in Presto functions to insert a new drop down box object, with the options listed above. The resulting program looks like this:

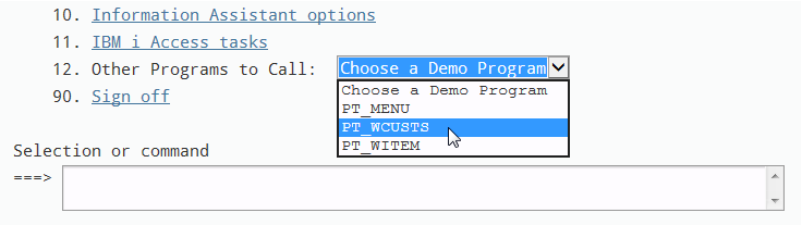


Figure 2-10. A new drop down list added to a green screen program.

In this illustration we are about to select one of the menu options.

This example uses a custom script to add something entirely new to the screen. To replace a text field with a drop down box you can easily use the visual editor to switch the element type from a 'Text Field' to a 'Drop Down Box'.

This is described in the section "Drop Down List" on page 226.

Adding an Event to a Drop Down Box

An alternate approach to adding this feature is to use the Events properties in the visual editor to automatically submit the call to the selected menu when the user selects an option from the list. Here is an illustration of how to add this feature (via the visual editor):

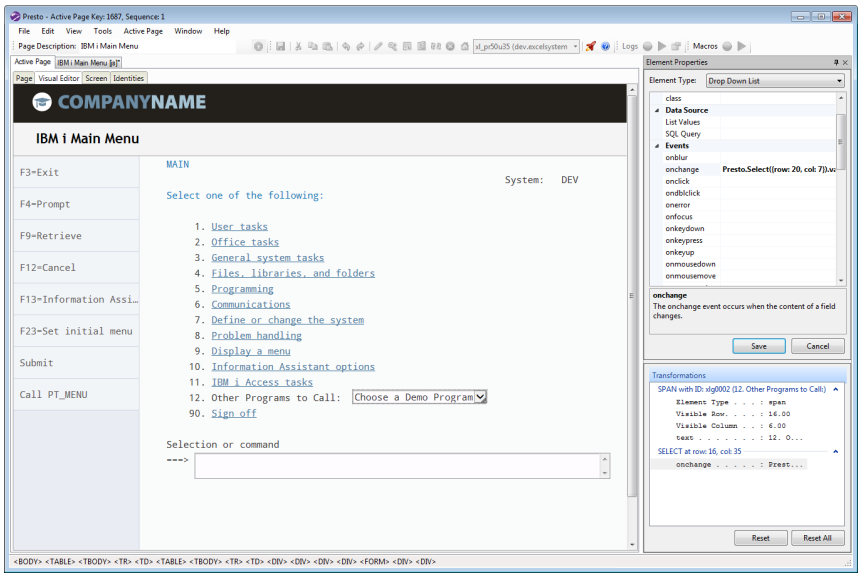


Figure 2-11. Adding an event to a drop down list.

In this illustration we have added:

```
Presto.Select({row: 20, col: 7}).val(this.value);SendRequest('EN');
```

to the onchange event, to automatically submit the call to the specified menu as soon as the option is selected from the list.

To test this, remove the 'onchange' property from the Presto.Insert transformation you added to the page originally:

```
onchange: "Presto.Select({row: 20, col: 7}).val(this.value);SendRequest('EN');"
```

Save your changes, and refresh the page to try it out.

Adding Features to Your Datepickers

The Visual Editor is the easiest way to add datepickers to your pages. This is described in the section “Date Picker Field” on page 223 and there is also an example in the tutorial. However, you can also add manual transformations to exploit more of the power and flexibility Presto’s jQuery date picker.

The sample applications shipped with Presto include a flat panel where you can edit your Order Header information:

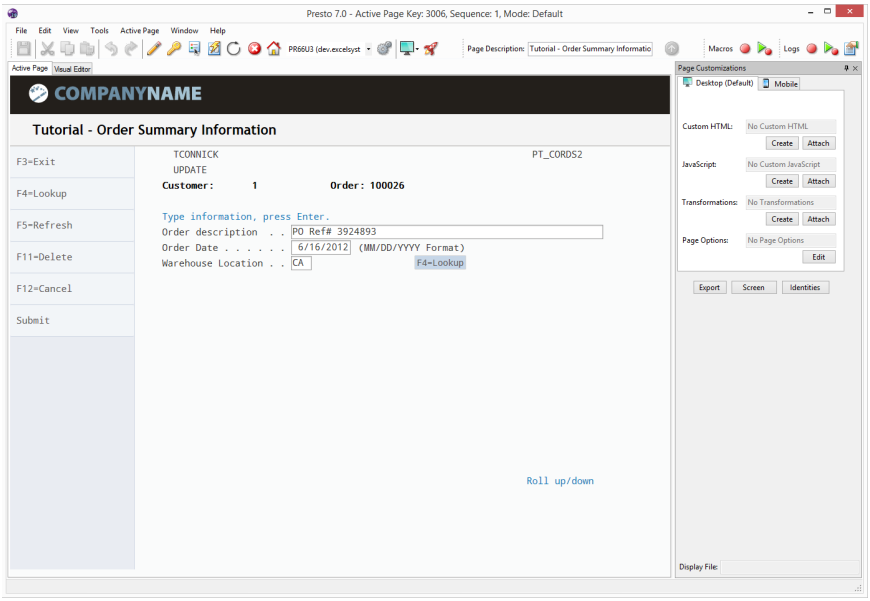


Figure 2-12. Editing the Order Header records.

As noted, we have worked with this panel previously in the tutorial and in other examples. In the illustration above we have removed all other customizations.

To get to this program call PT_MENU then take option 1. Select option 7=Orders for any customer, then use option 2=Change to edit an order.

Step 1: Create a Custom JavaScript for this page.

The option to create a new JavaScript page is at the right - click the Create button in this section to manually add the date picker transformation.

Step 2: Add your script.

This script will add a datepicker to the date field:

```
Presto.Transform({
        row: 7,
        col: 25,
        to_object: "datepicker",
"padding-right": "14px",
        background: "url('images/jscalendar.gif') no-repeat 100%"
});
```

Save your script.

The result in our running program looks something like this:

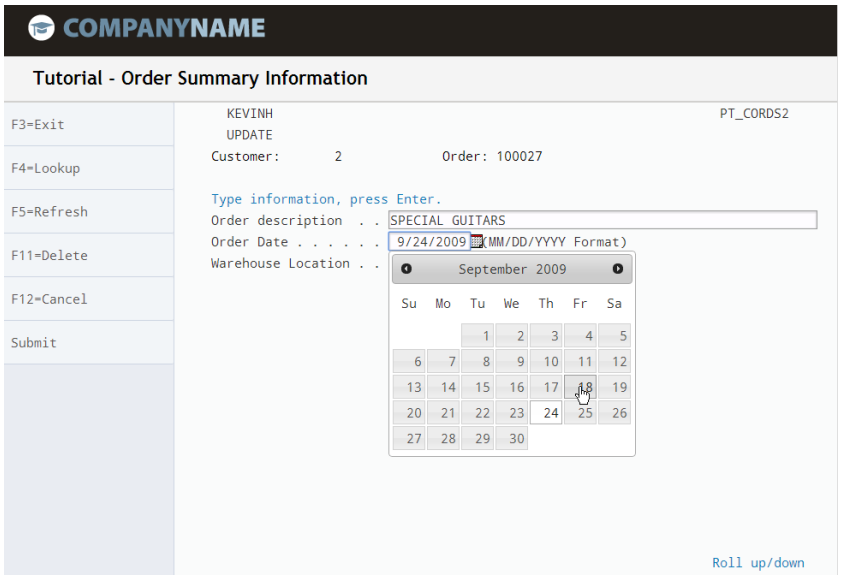


Figure 2-13. The date picker in the running program.

We have clicked the field and are about to pick a date from the calendar.

The date picker is a jQuery User Interface component. It is implemented as a method of the standard jQuery object (jQ). You can read more about it at:

http://docs.jquery.com/UI/API/1.8/Datepicker.

Additional Examples

There are many other options you can pass to date picker. Here are some:

```
// Present a calendar that advances a year at a time
Presto.Transform({
        row: 7,
        col: 25,
        to_object: "datepicker",
        stepMonths: 12
});

// Display two months at a time
Presto.Transform({
        row: 7,
        col: 25,
        to_object: "datepicker",
        numberOfMonths: 2
});

// Hardcoded switch to using another language for month and day names
Presto.Transform({
        row: 7,
        col: 25,
        to_object: "datepicker",
        monthNames: ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin',
'Juillet', 'Août', 'Septembre', 'Octobre', 'Novembre', 'Décembre'],
dayNamesMin: ['Lun', 'Mar', 'Mer', 'Jeu', 'Ven', 'Sam', 'Dim']
});

// Set first day of the week in Calendar View (Example picks Monday,
// alternative is to set it to 0 for Sunday)
Presto.Transform({
        row: 7,
```

```
        col: 25,
        to_object: "datepicker",
        firstDay: 1
});

//Allows you to change month & year through drop downs on picker
Presto.Transform({
        row: 7,
        col: 25,
        to_object: "datepicker",
changeMonth: true,
changeYear: true
});
```

Using a Script to Add Dynamic Images to a Page

The section "Using Screen Values to Dynamically Load Images" in the online tutorial starts with a broader discussion of some of the considerations to take into account when planning to incorporate images into a modernized site, and also has an example of how to do that using the visual editor. This section illustrates a similar approach using only a custom JavaScript.

Here is an illustration of the original screen where we'll be adding the image:

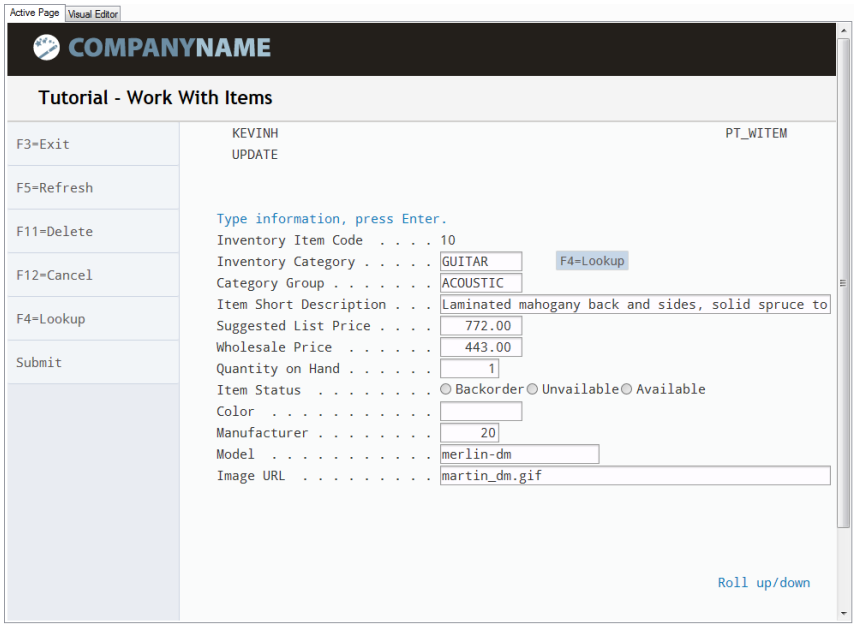


Figure 2-14. Editing item details, displaying only the fields.

Notice that this flat panel includes an Image URL field at the bottom. Hopefully your existing green screen applications already include a field this useful on the panel somewhere! Otherwise you will have to:

- find a way in your transformation code to uniquely identify an image based on other data that *is* on the screen, or
- add a field to your green screen program.

Another challenge is that in the current screen layout there isn't a particularly elegant location into which we can insert an image. We're going to take the easy route though, and just look at how to technically add the image and not worry about the layout, which is actually very easy to change in the visual editor.



It's worth reiterating at this point that most of what we're doing here is *web* application design, and Presto is merely your tool for marrying up your green screen application with the infinite possibilities of web applications. So with an objective like this one, there are many different ways to skin the proverbial cat - this example just describes one approach among many.

Step 1: Create a Custom JavaScript.

In Presto, navigate to the page above (option 2 from the menu PT_MENU, then option 2 to change a record).

Click the Create button to add a custom JavaScript to this screen.

Step 2: Load the Image

Add the following script to insert the new image:

```
Presto.Insert({
        vrow: "9",
        vcol: "80",
        object: "image",
        src: "prdemo/musicstore/<%=screen(17,31,50)%>",
        alt: "Image for: <%=screen(17,31,50)%>",
        border: 0
});
```

This bit of code relies on our having the appropriate Apache server mapping to retrieve the static file from the location indicated by the src parameter on the img tag: `'<img src="prdemo/musicstore/'`.

Our initial, functional result is this:

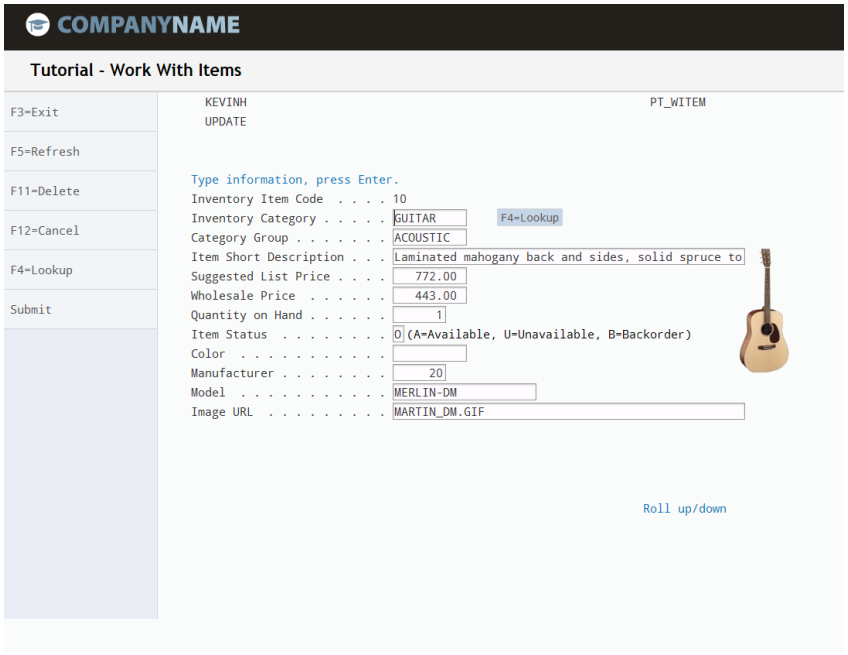


Figure 2-15. Inserting a dynamic image in your panel.

Not beautiful, but it works! As you roll through your list, the program will load the appropriate images.

Adding SmartCharts with Custom JavaScript

While the easiest way to add SmartCharts to your page is via the visual editor, you can also create them in custom JavaScript and HTML. Our tutorial application includes a program that lists items in the demo file MU_ITMF. Let's add a chart that summarizes quantities on hand. If you are following this example, you can add the chart to any page that has some white space on it.

Building a Dynamic Chart

Step 1: Add a SQL Query to the Presto Designer

Under the Tools menu in Presto click SQL Queries and add this query:

```
select imcat,sum(imqtonhd) from xl_prdemo/mu_itmf group by imcat
```

Name it SCQttyOnHand (the SC prefix indicates that the query is primarily intended to be used in a chart). Click OK to save and close the SQL query list.

Step 2: Add the SmartCharts™ script to a page.

At the list of Items click the option to create a custom JavaScript. For this example we have removed all other customizations from the screen.

This is the script

```
/* Chart Loading Dynamic Data Using Pre-Defined SQL */
Presto.Insert(
{
        vrow: "9",
        vcol: "2",
        id: "graph",
        to_object: "smartcharts",
        chart_type: "Column3D",
        sql_options: {
                name: "SCQttyOnHand"
        },
        chart_parameters: {
        caption: "Music Store - Quantity on Hand by Category",
        showvalues: "0",
        xaxisname: "Category",
        yaxisname: "Quantity On Hand"
        },
        width: 600,
        height: 250
}
);
```

We are adding this via the Custom JavaScript interface, so we don't need to include the `<script type="text/javascript">...</script>` tags.

Step 3: Save and refresh the page.

This chart is displaying dynamic data based on the SQL query we defined. This is how the chart looks when it runs:

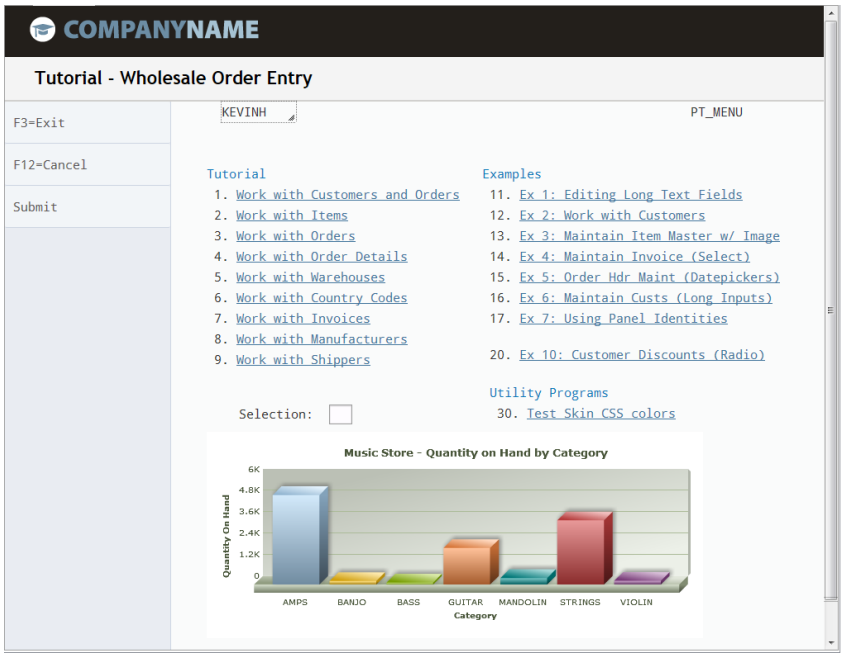


Figure 2-16. SmartCharts™ in Presto.

To accommodate this on the page we moved a few other elements around, making more space at the bottom.

You can select the chart and adjust its attributes in the visual editor, or add further properties in the manual transformation.

Building a Chart Using Static Data

You can also use a custom script to create a chart that displays a set of static values (defined in your transformation code directly).

Here is some sample code:

```
/* Chart demonstrating static data */
Presto.Insert(
{
        vrow: "3",
        vcol : "42",
        id: "graph",
        to_object: "smartcharts",
        chart_type: "Pie3D",
        options: {
"Guitar": 554,
"Drums": 403,
"Bass": 155,
"Violin": 80
        },
        chart_parameters: {
        caption: "Instruments popularity",
        showPercentageValues:"1"
        },
        width: 300,
        height: 260
}
);
```

Here is an illustration:

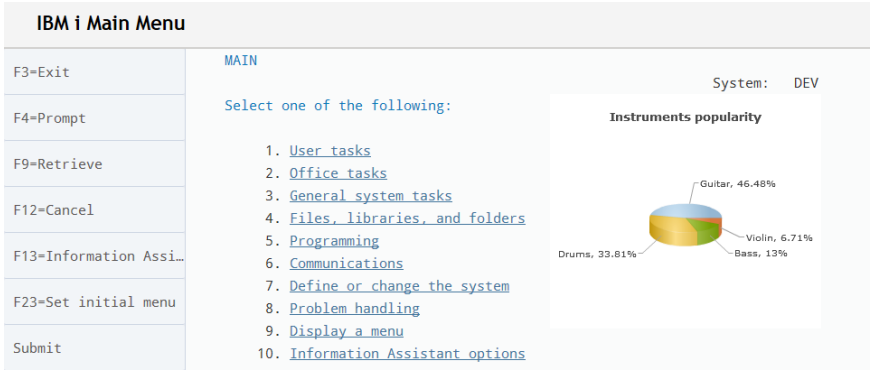


Figure 2-17. Generating a static chart.

Static graphs are less useful than dynamic ones, but may be appropriate for certain situations.

Building a Chart Using Screen Values

You can also combine a static graph structure with dynamic data like this:

```
Presto.Insert(
{
        vrow: "9",
        vcol : "78",
        id: "graph",
        to_object: "smartcharts",
        chart_type: "Pie3D",
        options: {
"Guitar": <%=screen(10,5,2)%>,
"Drums": <%=screen(11,5,2)%>,
"Violin": <%=screen(13,5,2)%>
        },
        chart_parameters: {
        caption: "Instruments popularity",
        showPercentageValues:"1"
        },
        width: 300,
        height: 300
}
);
```

Note that in this instance we have replaced the quantity on hand values with values retrieved from the screen. Our demo program doesn't include the necessary data, but you can use this technique in your own applications.

Support for Retrieving Dynamic Values from the Screen

Presto includes two functions you can use to access raw screen data. There are a variety of applications for these functions, but typically you are grabbing screen values to use in other custom code.

- Presto.GetRawScreen(AjaxParams). Expects an object AjaxParams which has to implement "success" method.
- Presto.GetScreenValue(nRow, nCol, nLen). Returns the text from a desired location of a screen. It has to be called in a "success" of Presto.GetRawScreen otherwise it may not return expected data.

Here is an example of the first function:

```
Presto.GetRawScreen({
        success: function (s)
        {
           var sScreenValue = Presto.GetScreenValue(6, 3, 33);
           alert(sScreenValue);
        }
});
```

When we add this to the first tutorial menu, this is the result:

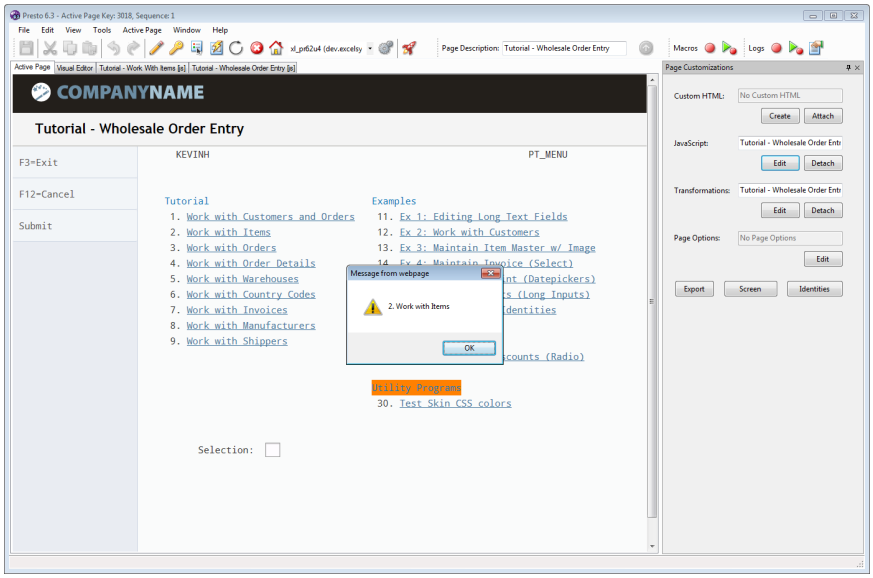


Figure 2-18. Using the GetRawScreen function to retrieve a screen value.

This example has grabbed a chunk of screen text and displayed it in an Alert.

Here is another more elaborate result, combining various snippets from the screen:

```
Presto.GetRawScreen(
{
        success: function()
        {
                var sScreenValue;
```

```
                var sScreenValue1;
                var sScreenValue2;
                for (var i=0; i < 10; i++)
                        {
                        sScreenValue = Presto.GetScreenValue(5,2,28);
                        sScreenValue1 = Presto.GetScreenValue(6,3,33);
                        sScreenValue2 = Presto.GetScreenValue(7,3,18);
                        }
        alert(sScreenValue +" " +sScreenValue1 +" "+sScreenValue2);
        }
});
```

On the same tutorial menu above, this outputs:

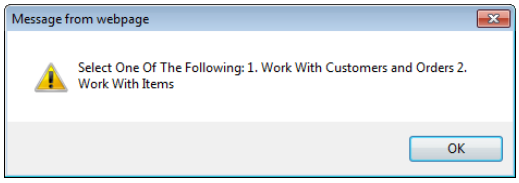


Figure 2-19. Using the GetRawScreen and GetScreenValue functions.

These are just simple examples illustrating the principle, but hopefully you can see how you could use these in your own applications to retrieve screen values for further processing, or to control additional options.



These techniques can also be implemented in Custom HTML, by just wrapping the example code above in

```
<script type="text/javascript"> ... </script>
```

tags.

Enhancing Autocompletes Manually

The easiest way to add autocompletes to your application is using the visual editor. However, this feature is implemented via jQuery and if you are a more experienced web developer (or have demanding users!) you may want to take advantage of some of the additional features available via jQuery and the jQuery UI that aren't [currently] implemented as properties in the visual editor.

You can find information about the jQuery autocomplete, and customizations you can implement using the jQuery UI, at this page:

http://jqueryui.com/demos/autocomplete/

For example, the visual editor includes a Minimum Length option for the autocomplete, but you may instead prefer to use the delay option.

Below are a few examples you can implement via Custom Scripts (or in Custom HTML).

Using an Array as the Source for an Autocomplete

This example uses an array as the source for the autocomplete:

```
Presto.Transform({
        row: 6,
        col: 53,
        to_object: "autocomplete",
        source: ["fred", "larry", "alex", "robert", "mary"]
});
```

The items listed are the suggestions provided to the user.

Using an SQL Query's Result Set as Source

Here is how to manually create an autocomplete that uses the result set from an SQL query to build the list:

```
Presto.Transform({
        row: 6,
        col: 53,
        to_object: "autocomplete",
        source: {
                name: "users2"}
});
```

In this case the query named "users2" will be executed once and its result set used as a source for auto suggestions.

Using an SQL Query with the Parameter Name of the Search Term

You can also build an autocomplete that uses the result set from an SQL query, with a parameter:

```
Presto.Transform({
        row: 6,
        col: 53,
        to_object: "autocomplete",
        source: {
                name: "users1",
                param: "user"}
});
```

In this case the query with the name "users1" and the parameter "user" (which will contain the search term) will be executed every time you type into the field at row 6, column 53.



There is a functional difference between queries using a dynamic search terms and those that do not. When using queries with no search term, the full results table is loaded when the screen is loaded and Presto uses JavaScript to find matches within the set. For queries with a search term a new http request is submitted (and the list reloaded from scratch) every time a character is typed.

Contact Technical Support if you have questions.

Using an External Web Program or Service to get Data for a Drop Down Box

In earlier sections of this guide we looked at examples of adding static and dynamic drop down lists to your pages. This section illustrates a more powerful customization technique using an Ajax call to load a drop down list with the results of an SQL query. While there are simpler ways in Presto to generate simple drop down lists, the general technique of invoking an external program via a script can be useful in other situations, and this example is included here to foster further ideas for you!

Overview

AJAX lets you make calls to the server from within the existing page and dynamically change the content of the page as the user interacts with it. In this example we are replacing a typical 5250 green screen coding pattern with one that is more like a native web application. In the green screen app, users can type a Warehouse code for an item or position the cursor on the Warehouse field and press F4 to present a Window lookup to find it. Let's replace that with a drop down box by calling a simple WebSmart program. The advantages of this are:

1. The user doesn't have to worry about positioning the cursor. They can use the mouse.
2. The user sees all available options by clicking on the drop down.
3. We eliminate the need for the user to press a Function key.
4. We avoid making unnecessary calls to the server to validate the warehouse, since it is now impossible for the user to select (or type) a warehouse code that is not in the warehouse master file.

Here's how the screen looks before we implement our drop down:

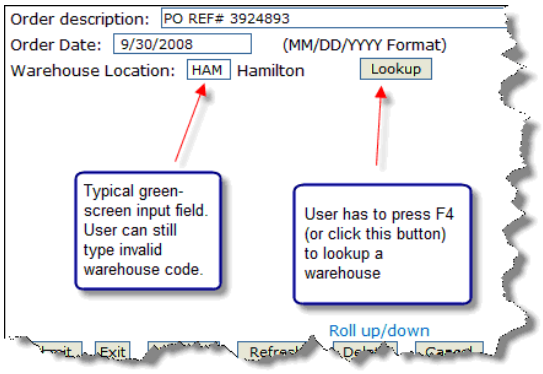


Figure 2-20. A typical green screen approach, rendered without change in a web page.

After making our changes, the page will work like this:

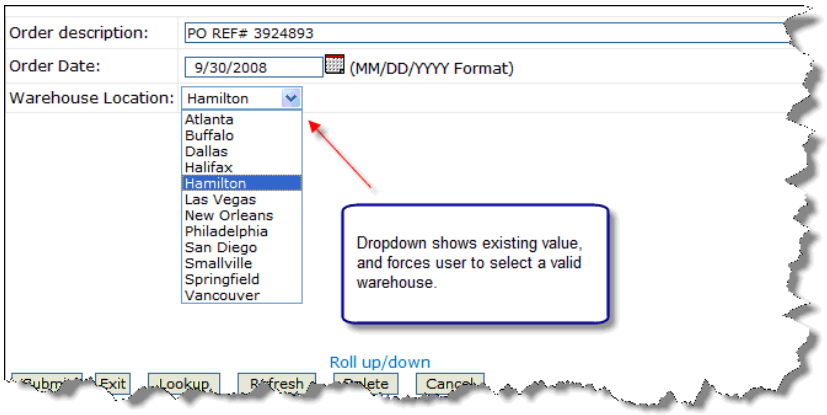


Figure 2-21. Using an Ajax function call to load a dynamic list of options.

To implement this feature, we took three steps.

Step 1: Create a Program to Build the JSON string for the Presto.Transform function.

Our first step was to create a WebSmart program that builds the HTML for our drop down box. The PML code for the program looks like this:

```
func Main()
{
        // Program Name:PSAJWHSED2
        // Program Title:Presto AJAX Warehouse
        // Purpose: Return a JSON object to construct values for a
        // drop down. Called via an AJAX call from a Presto.Transform
        // JavaScript object.

        setlibl("*FILES");
        openfile("*ALL");

        // Write a the values of select box as a JSON object
        runtask("default", jsonselect);
}

// Build an object using JSON notation that meets the requirements of
// the options attribute in the Presto.transform function.
// For example: Presto.Transform({row: 8, col: 25, to_object:
// "select", value: "<%=screen(8,25,3)%>",
// options: "/prestodemo/psajwhsed2.pgm" } );
// The value for options might be: [{"value":"ATL","text":"Atlanta"},
// {"value":"BUF"," text":"Buffalo"},{"value":"DAL"," text":"Dallas"}]

func jsonselect()
{
        crtfld(dquote, 2, "A", 0, "double quote");
        crtfld(value, 12, "A", 0, "double quote");
        crtfld(text, 11, "A", 0, "double quote");
        dquote = "\"";
        value = "\"value\" : ";
        text  = "\"text\" : ";

        wrthtml("[");   // Start the JSON object
        posrcd(MU_WHSF, "*FIRST");   // Start of warehouse file
        getnxtrcd(MU_WHSF, "*NOKEY");  // Read first warehouse code
        while(MU_WHSF.status == RecordRetrieved)
        {
                wrthtml("{"); // open the JSON object
```

```
                wrthtml(value,  dquote, MWWHCD, dquote);
                wrthtml(",");
                wrthtml(text,  dquote, MWWHDS, dquote);
                wrthtml("}"); // End the JSON object
                getnxtrcd(MU_WHSF, "*NOKEY");  // Read next warehouse code
                if(MU_WHSF.status == RecordRetrieved)
                {
                        wrthtml(",");
                        // Another code to write - insert a comma before the
                        // next name/value pair.
                }
        }
        wrthtml("]"); // End the array
}
```

While you don't have to use a WebSmart program specifically to generate this, the code above should give you a sense of what you need to do.

Step 2: Populate the Drop Down Box

Our final step is to add this code to populate the drop down box. You can add this as custom JavaScript to the page:

```
Presto.Transform({
        row: 8,
        col: 25,
        to_object: "select",
        value: "<%=screen(8,25,3)%>",
        options: "/prestodemo/psajwhsed2.pgm"
});
```

This Presto.Transform function replaces the object found at position 8,25 with a drop down box, where the options are returned from the call to the program identified above.

The program is designed to read through the file, and return the necessary records in the format required by the transformation:

```
[{"value":"ATL","text":"Atlanta"}, {"value":"BUF","text":"Buffalo"},
{"value":"DAL","text":"Dallas"}, {"value":"HAL","text":"Halifax"},
{"value":"HAM","text","Hamilton"}, {"value":"LAS","text":"Las Vegas"},
{"value":"NOL","text":"New Orleans"}, {"value":"PHI","text":"Philadelphia"},
{"value":"SAN","text":"San Diego"},{"value":"SMA","text":"Smallville"},
{"value":"SPR","text":"Springfield"},{"value":"VAN","text":"Vancouver"}}]
```

Step 3: Remove Ancillary Objects

Because you're handling this feature differently, you can get rid of the extra text and objects (the lookup button) on the screen using a couple more lines like these (or you can do this via the visual editor as well):

```
Presto.Transform({
        row: 8,
        col: 47,
        display: "none"
});

Presto.Transform({
        row: 8,
        col: 29,
        display: "none"
});
```

Because the lookup button is generated to refer back to the field, when you transform the field into a drop down, the transformation also picks up the button. And because you are pulling the warehouse descriptions right into the drop down list, you no longer need to display them beside it.

Using the File custom.js

When you install Presto the IFS files for the software includes the folder userresources/ in the directory esdi/presto. In that folder you can find custom.js. This JavaScript file is loaded by presto.js once at the beginning of each Presto session. It is NOT reloaded as users move from one page to another within the site, so you don't need to be unduly concerned about its size. Use custom.js to implement changes that will affect your entire site.

Datepicker Localization

The default datepicker that you can implement via the Presto visual editor matches the typical settings for a North American audience. If you are building your application for use in another country, you can take advantage of a jQuery data localization feature. Your custom.js file includes a description of the change to make and a snippet of code you can uncomment to switch your datepicker's country settings.

Enabling an Alarm Sound

In custom.js you will also find some example jQuery you can uncomment in order to enable an alarm sound in your browser.

Turning Off the Message Waiting Indicator

In this version of Presto a new Message Waiting indicator is shown if the message queue is not empty. Note that once DSPMSG is called the indicator disappears for the active session (even if the queue is not empty). It will reappear for a new session.

You can turn this feature off by adding the following code to your custom.js file:

```
jQ(document).unbind("TurnOnMessageLight");
```

Suggested Structure for Additional custom.js Enhancements

To use custom.js to add custom features to your entire site we suggest you start with the template in our online Knowledge Base article 8923:

https://www.myfreschesolutions.com/kb?kbnum=8923

You can also find instructions in section "Applying a Transformation to Every Page" in the online tutorial.

Add Key Mapping

Presto allows you to change your keyboard mapping by using the AddKeyMap Javascript API.

You can add code similiar to this to your custom.js:

```
AddKeyMap({nFromKey: 17, which: 13, ctrlKey: false});
```

This example maps the CTRL key (17) to the Enter key (13).

AddKeyMap expects nFromKey as a code of the key that is going to be mapped to another key.

The API can overwrite any attribute of JavaScript event object in order to achieve the desired outcome, like for example: ctrlKey, altKey, shiftKey, which, keyCode, charCode

```
AddKeyMap({nFromKey: 13, nToKey: 107, charCode: '+'.charCodeAt(0)});
```

This example is mapping the ENTER key (13) to the Number Pad + (107 interpreted as Field Exit).

Note that browsers can be inconsistent about which keycodes they use, so be sure to test in all the browsers you plan to use.

To see a list of common JavaScript key code, please refer to one of the following sites:

```
http://www.cambiaresearch.com/articles/15/javascript-char-codes-key-codes
https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent#Virtual_key_codes
```

Using a Transformation to Upload Files

If you want to create a button that will allow users to upload files to IFS there is a transformation object that you can use.

The object type is called: fileupload

It'll require using JavaScript and the Presto.Insert() or Presto.Transform() command.(See the appendix in the Userguide for more information on these.)

It will upload the users document to the IFS into the folder: /esdi/xl_presto/uploads

Where xl_presto is the name of your Presto environment. Here is an example that could be placed in Page Level JavaScript:

```
jQuery(document).one("PageComplete", function()
{
        Presto.Insert(
                {
                "object": "fileupload",
                "value": "Upload File",
                "vrow": "10",
                "vcol": "20",
                "events":
                {
                        "done": function() {alert('File uploaded successfully.');},
                        "fail": function() {alert('File upload error.');}
                }
        });
});
```

Note: for security reasons you must specify which types of documents Presto will allow to be uploaded to the server. Follow Knowledge Base article 16384 for instructions:

https://www.myfreschesolutions.com/kb?kbnum=16384

Retrieve Job Info

Presto provides three JavaScript functions to retrieve info about the Presto job.

GetUser

Get the user of the interactive job.

```
Presto.GetUser(function(result) {
   // result is a string.
   console.log(result);
})
```

Considerations for Use:

- Presto.GetUser returns empty string on the Sign On screen as there is no valid user yet.

GetJob

Get the job information for the interactive job.

```
Presto.GetJob(function(result) {
   console.log(result);
})
```

Considerations for Use:

- Presto.GetJob returns empty string on the Sign On screen as there is no dedicated interactive job yet.
- The job name returned by Presto.GetJob is of the form "QPADEV001KCHRIS 410700". This contains the device name, user name, and job number for the interactive job. This is only available once the user has signed in. The device name and user name portions are both padded to 10 characters. The job number is padded to 8 characters. Total length of 28 characters. The padding is on the left for all 3 segments.

GetDeviceName

Get the Device name for the interactive job.

```
Presto.GetDeviceName(function(result) {
   console.log(result);
})
```

Considerations for Use:

- Presto.GetDeviceName will return the device name in use from the Sign On screen as that device is already reserved for the current session.

SentRequest Event

The SentRequest event is triggered when a page request is in progress, after a SendRequest event. SentRequest indicates that a page request has been sent to the server. This allows you to execute custom JavaScript code after a request to the server has been sent. Here is an example:

```
jQuery(document).on("SentRequest", function(){
   console.log("Sent Request!!");
});
```

For example, the above code executes for each page navigation - click, keystroke, or SysReq key.

Considerations:

- The SendRequest event, in contrast to SentRequest, is triggered when a request is about to be sent and gives the opportunity to modify or cancel that request. The SentRequest event notifies that the request has been sent to the server. Unlike SendRequest, this event cannot be cancelled.

PlayMacro Event

The PlayMacro event is triggered before a macro is played. Here is an example:

```
jQuery(document).on("PlayMacro", function(){
   console.log("Macro Played");
});
```

This is useful for running custom JavaScript code on the page navigation

CHAPTER 14

Chapter 14 - Creating and Using Custom HTML

In previous chapters we have reviewed the customization tools available from the Active Page and Visual Editor. The options we have discussed include:

- various settings and features available via the Page Properties, including table, window, function key and menu link settings
- myriad transformation options to change the properties or type of your screen elements via the visual editor
- adding custom scripts to individual pages or your site as a whole

Custom HTML complements the tools above by giving you yet another level of power over your output: control over all the HTML generated for a page.

This chapter describes how to build custom HTML. The tutorial section of this guide also includes examples you can refer to for additional guidance.



A key concept to internalize as you start adding advanced custom features to your applications is that you are mainly doing Web, rather than Presto, application development.

So while the Fresche support team can help you with many of the questions you may have, you can also find many tips and techniques via general web searches. The best web development resource is the web itself!

Creating New Custom HTML Entries

To begin creating custom HTML open the Presto Designer and go to the page you want to work with. Then click the 'Create' button in the Active Panel options:

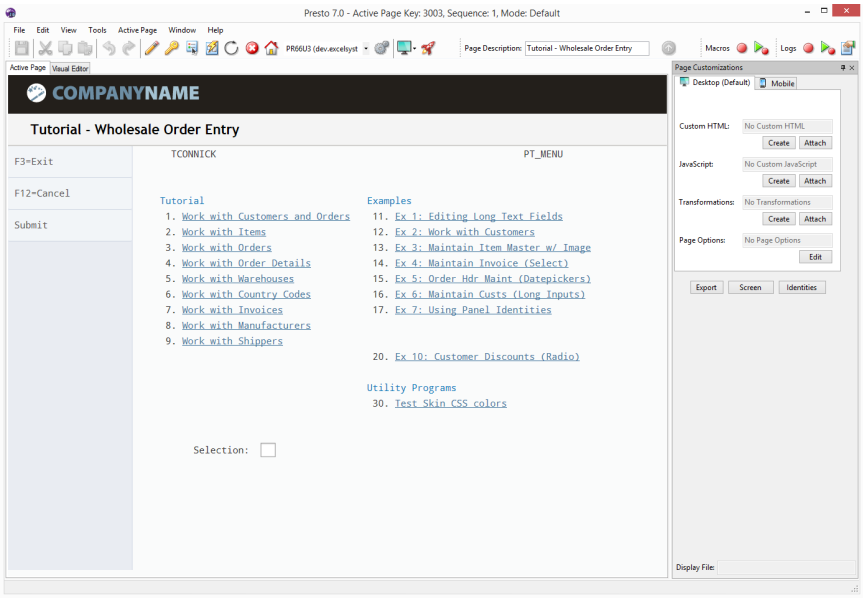


Figure 2-1. Preparing to create custom HTML.

The 'Create' button is part of your Page Customizations options on the right.

When you begin working with the HTML, the Presto Designer switches to this mode:

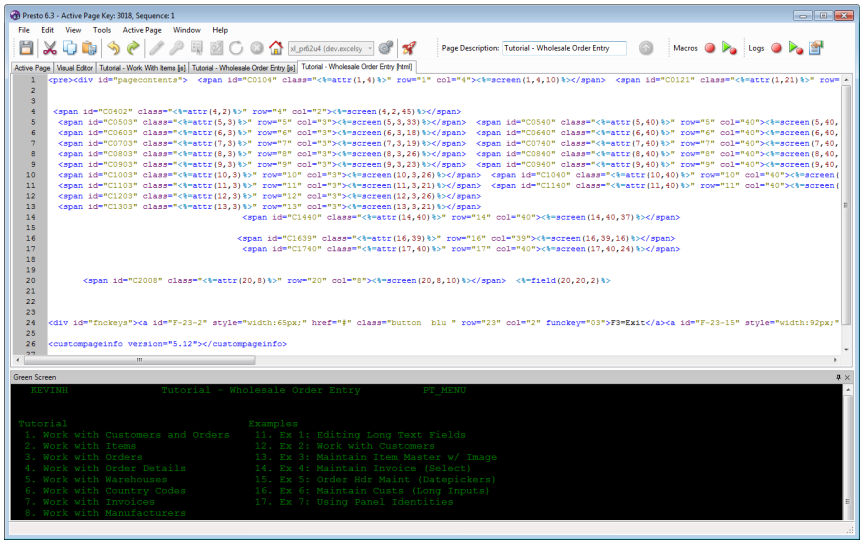


Figure 2-2. Edit mode for custom HTML.

Notice the following:

- In the main area on the top you can see the code that is streamed to the browser for this page.
- The tab above this area displays the page title.

- On the bottom you can see what your original screen looks like. (If this area isn't initially visible, you can display it from the View>Screen menu option.)

  You will also use this area when you need to replace static sections of HTML with dynamic field values. More on this later!
- Use the toolbar icons to save/copy/paste/delete and perform other operations related to your open file.

Use the Find/Replace tools to locate text strings in your HTML and optionally replace them with others. You can find a complete list of the Find/Replace options and shortcut keys under the Edit menu.

The next few pages review the general structure of the generated HTML before revisiting some of the key concepts in more depth.

The Apprentice's Guide to Editing the HTML

Customizing Presto pages is like working with any web page. You can build functional web sites using a half dozen HTML tags, but with some knowledge of more advanced techniques you can also create very sophisticated designs.



A key concept to internalize as you start adding custom features to your applications is that what you are doing is fundamentally just web application development. When you start adding sophisticated web features via custom HTML Presto essentially becomes a vehicle to facilitate delivery of your IBM i data stream, over which you then add your customizations at will.

What this means is that when you have questions, the Fresche Technical Support will have answers, but you can also find myriad tips and techniques via general web searches and by visiting jQuery and JavaScript sites and forums (as we do ourselves!).

Review the Basic HTML

Let's begin by looking at some samples of the HTML generated by Presto for the menu we started editing in Figure 2-2.

Here is the HTML generated for the non-blank fields in the first line of the program (we have inserted some line breaks to make it more readable in this document):

```
<pre><div id="pagecontents">
<span class="<%=attr(1,4)%>" id="C0102" style="    " row="1" col="4">LARRY </span>
<span class="<%=attr(1,21)%>" id="C0121" style="  display: none; " row="1"
col="21">Tutorial - Wholesale Order Entry    </span>
<span class="<%=attr(1,67)%>" id="C0167" style="    " row="1" col="67">PT_MENU    </span>
```

The following sections discuss the main characteristics of this part of your HTML.

Where are the page-level HTML tags?

If you already know a little about HTML you may notice that the editable HTML shown in Presto doesn't include the standard tags that normally frame HTML pages: `<html>`, `<body>`, `<head>` etc.

This essential section of your page is pulled in at run-time from the skin files in your IFS (normally /esdi/XL_PRESTO/templates/xl_layout.html). These are applied to every page in your environment in order to give your site a consistent framework.

This CSS tag allows you to set broad attributes in the style sheet that affect the entire page output. This `<div>` tag is closed near the bottom of your page, just before the error message field.

Text Strings

Within your page you'll see various instances of text strings that look something like this one:

```
<span class="<%=attr(1,67)%>" id="C0167" style=" " col="67" row="1">PT_MENU </span>
```

The `<span>...</span>` tags are used to add attributes to text strings.



Wherever you see a 'class' attribute within the span tag it means your page is referencing characteristics defined in the site skin (style sheet). See , Working with Your Presto Skins, for further information.

Within the span tags, there are these components:

- The span ***id*** allows you to uniquely reference individual elements of your page. In this case, the span shown above is wrapped around the program title. The section below discusses this attribute.
- The span ***class*** tag is linked to a style sheet definition of the attribute. The class is dynamically picked up from the attribute byte on your actual green screen program, every time the program runs.
- The ***row*** and ***column*** attributes allow the program to keep track of where the field or text is located on the screen, and where the cursor is. For example, the system `WRKACTJOB` page allows you to sort the list by the column where your cursor is.

Every Field has a Unique ID!

A key concept shared by most of our examples is that your custom code must somehow know what panel element to apply the transformation to. If you are using JavaScript to replace a text field with a drop down box, how do you tell your script which field to replace?

When Presto generates the HTML for a given panel at run time, every element is assigned a unique ID based on its original position on the panel.



This also means that if you make changes to your original green screen application, it may seem like your Presto customizations are 'broken'. In fact, they probably just need to be re-attached to the new panel location.

Text Input Fields

The menu shown on the previous page includes one input-capable field (text field), which is rendered on the screen using this HTML code:

```
<span class="<%=attr(20,8)%>" id="C2008" style=" " col="8" row="20">Selection:</span>
<%=field(20,20,2)%>
```

The field itself (at the very end of the line above) is configured to both dynamically render any data on the screen at the specified positions, and accept input.

Function Keys

Near the bottom of your page you will have one or two rows of function keys that look something like this example:

```
<div id="fnckeys">
<a class="button blu " id="F-23-2" style=" width: 65px; " col="2" row="23"
funckey="03">F3=Exit</a>
```

This code sets the attributes for your function key buttons (referencing your skin's style sheet) and the text value shown on the button (`value="Submit"`).

Notice especially these sections (one for each function key):

```
onclick="SendRequest('EN');"
onclick="SendRequest('03');"
onclick="SendRequest('12');"
```

When the user clicks these buttons, the page executes a SendRequest, specifying the function key pressed. Here is a table of the valid values for your keys:

ID	Description
01 - 24	Functions keys 1-24
EN	Enter
RU	Roll Up
RD	Roll Down

Error Message Subfile

The last element you'll see on most of your pages is the code to handle presenting your error messages:

```
<input class="ErrorMsg" id="ErrMsg" style="  " type="text" size="79" maxlength="79"
readonly="true" value="<%=trim(screen(24,1,80))%>" col="1" row="24" fieldsize="79">
```

You shouldn't normally have to touch this.



The potential gotcha here is that Presto by default assumes that you have line 24 reserved for error messages. If instead you have placed text or fields in that row, you won't initially see them in Presto because Presto is expecting to display messages there.

You can address this using the 'No Message Subfile' option. You can set this option at the environment level, or just for individual pages.

Other Basic Panel Elements

In addition to the elements discussed above, there are a few others that you'll often see.

Command Lines

Here is how an IBM i command line typically looks in your HTML (here including the text before it):

```
<span id="C1902" class="<%=attr(19,2)%>" style="   " row="19" col="2">Selection or
command    </span> ...

<span id="C2002" class="<%=attr(20,2)%>" style="   " row="20" col="2">===&gt;    </span>
<%=field(20,7,153)%>
```

The pseudo-HTML tag `<%=field(20,7,153)%>` maps the section of the original green screen program (starting on line 20 column 7, for 153 characters) to this field on the web page.

Option Fields

When you create custom HTML over a subfile list program (where you have defined a table around the list) a typical row might look like this:

```
<tr class="Row2">
```

```
<td> <span id="C1202" class="<%=attr(12,2)%>" row="12" col="2" onmouseup="SavCurPos
(this);"> </span><%=field(12,3,1)%>
<span id="C1205" class="<%=attr(12,5)%>" row="12" col="5" onmouseup="SavCurPos
(this);"></span></td>

<td><span id="C1206" class="<%=attr(12,6)%>" row="12" col="6" onmouseup="SavCurPos
(this);"><%=screen(12,5,27)%></span></td>

<td><span id="C1232" class="<%=attr(12,32)%>" row="12" col="32" onmouseup="SavCurPos
(this);"><%=screen(12,32,7)%></span></td>

<td><span id="C1240" class="<%=attr(12,40)%>" row="12" col="40" onmouseup="SavCurPos
(this);"><%=screen(12,39,20)%></span></td>

</tr>
```

In this block of HTML, the first column of the row (within the first set of `<td> ... </td>` tags) includes the input-capable option field. The remaining columns are pulling in variable field information from the positions referenced in the green screen program.

Subfile Records

In a subfile, the table cells for the variable portions vary only in the row (the first block references data on row 9, the second block references data on row 10). And if you look at the first table cell on each, you can see that difference in the value parameter for the subfile option fields: `<%=field(9,3,1)%>` and `<%=field(10,3,1)%>`.

The Presto Screen, Field and Attr [Attribute] Tags

When you create and work with Presto custom HTML you are working directly with HTML. While most of the HTML you'll see in Presto is standard, there are three tags in particular that are unique to Presto. The tags are these:

- <%=screen(x,y,z[trim])%>
- <%=field(x,y,z)%>
- <%=attr(x,y)%>

In each of these tags, X and Y correspond to a screen position in the original green screen program. The literal Z is the length of the text string or field found at the specified X and Y coordinates.

Presto uses the screen, field and attr tags to dynamically render screen text, fields, and attributes. It is important that you understand what these tags are and how to use them because they give you control over some of the most important aspects of your page content.

Consider this section of a page in our tutorial application:

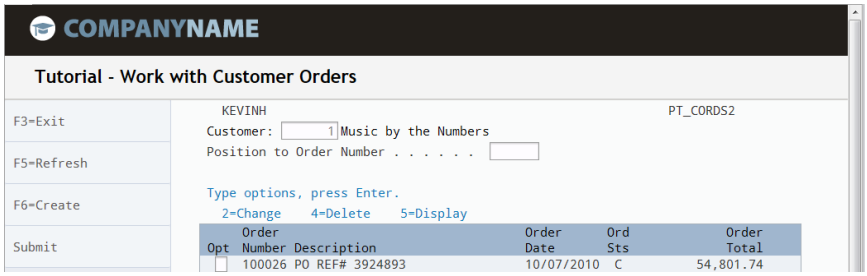


Figure 2-3. Sample page displayed in the Presto Designer.

In this illustration, notice the four types of content below the header:

- **Static Text.** The program title and name displayed in the first line never change in the original program.
- **Variable text.** The Customer Number [1] and Name [Music by the Numbers] can vary depending on the values selected by the user in the calling program.
- **Field Attributes.** While not obvious at this document's resolution, the Customer Number and Customer Name fields are in a bold typeface.
- **Input Capable Fields** The program above allows the user to enter an Order Number to position the list.

Presto generates unique HTML for each of these types of content.

Static Text

This is the HTML that the program above uses to display the program title (broken into sections here to make it clearer):

```
<span class="<%=attr(1,23)%>" id="C0123" style="  display: none; " col="23"
row="1">Tutorial - Work with Customer Orders</span>
```

You can read the program title in the HTML because it is rendered as static text. No matter where this program is called from or what parameters are passed to it, this title won't change in the web version of this page.

Notice that this is also set to non-display, because there is a skin-level script that grabs the title from its original location and puts it on its own line.

Dynamic Text

When Presto identifies that a text string on a panel is in fact a field (which is impossible to do with 100% accuracy for output-only fields) it creates pseudo-HTML tags for those. In the program illustrated above, there are two examples of this:

```
<%=screen(2,12,8)%>
<%=screen(2,20,60)%>
```

These tags tell us that at the spot where we find them in the HTML Presto is dynamically displaying whatever the green screen program currently shows at

- the 8 characters starting at position 2, 12; and at
- the 60 characters starting at position 2, 20.

These values correspond to the Customer Number and Name, respectively.

What this means is that if this program is called with a different customer number as a parameter, the web version will reflect that - as you would expect.



Keep in mind that while your custom HTML's initial layout puts the <screen> tag at the same position where Presto found the content in the original program, you don't have to leave it there!

This is a key concept in revamping your page layouts in preparation for display in a browser. You are no longer constrained to the original page's layout because you can move your <screen> tags anywhere in the page, and regardless of their location, they retain their link to the data's location in the original green screen program.

Text and Field Attributes

In your HTML, Presto dynamically pulls in the original program's display attributes for every field and text string on your page. As an illustration of this, consider the Customer Number's screen value:

```
<span id="C0212" class="<%=attr(2,12)%>" row="2" col="12"> <%=screen(2,12,8)%> </span>
```

In this section the with an ID of C0212 (the element's coordinates) containing the screen value at position 2,12 is also assigned a CSS class derived from the attributes at that same position, like this:

```
class="<%=attr(2,12)%>"
```

The CSS files (skins) shipped with Presto include classes for each of the attribute combinations supported on the 5250 display.

Sample Green-Screen Attribute to CSS Tag Cross Reference

Here is a sample table of how one skin renders the attributes picked up from the green screen programs in the CSS:

Description	Class Tag	Attributes
Underline	.ul	text-decoration: underscore;
Green	.grn	color: #00FF00;
Green/Reverse Image	.grnr	color: black; background-color: #00FF00;
Green/Reverse image	.rev	color: black; background-color: #00FF00;
Green/Underscore/reverse	.ur	color: red; text-decoration: underscore;
Green window border	.winbordergrn	border: 2px solid black !important;
Red	.red	color: #FF0000;
Red/Reverse image	.rdr	color: black; background-color: #FF0000;
Red/Underscore	.rdu	color: #FF0000; text-decoration: underscore;

These may vary depending on the skin you're using. To review the complete set open the file StyleSheeet.css in your skin folder.

Input Capable Fields

The last unique element in the screen sample illustrated above (Figure 2-3) is the input capable position-to field. In your HTML, this field looks like this:

```
<%=field(3,40,6)%>
```

The field tag is unique to Presto, and in the web page it generates a text box with attributes matching those of the field found at that location in the 5250 display.

Using Screen, Field and Attribute Tags

You can use screen, field and attribute tags in your JavaScript or jQuery functions. You can also replace the row column locations and length attributes within the tags with variables to make the tags dynamic (even more dynamic than they already are!).

For examples of how to use these tags please review the Presto examples and How-To Knowledge Base articles at our web site.

Support for Trimming Screen Values

You can also trim screen values. This feature is particularly useful when incorporating screen values into custom options and custom code.

Here is an example of how to trim a screen value:

```
<%=screen(1,36,45,trim)%>
```

The default (no trim) looks like this:

```
<%=screen(1,36,45)%>
```



This feature is also useful for building links and image tags!

Generating Dynamic Screen Tags for All Text Constants

When you create a new Custom HTML page Presto has traditionally generated dynamic tags for input capable elements, but rendered text constants as-is. However, screens sometimes include text constants that change every time the program is run. Presto may not be able to automatically detect this based on the single screen you are converting, and developers have typically handled this situation by manually replacing these apparent constants with <screen> tags.

This release includes a new Advanced Configuration option you can turn on to have Presto build dynamic screen tags for every text constant.

As an illustration, with this option turned off, your custom HTML page would look like this:

```
<span id="C0248" class="<%=attr(2,48)%>" row="2" col="48">System  . . . . . :</span>
<span id="C0270" class="<%=attr(2,70)%>" row="2" col="70">EDEV    </span>
<span id="C0348" class="<%=attr(3,48)%>" row="3" col="48">Subsystem . . . . :</span>
<span id="C0370" class="<%=attr(3,70)%>" row="3" col="70">QINTER    </span>
<span id="C0448" class="<%=attr(4,48)%>" row="4" col="48">Display . . . . . :</span>
<span id="C0470" class="<%=attr(4,70)%>" row="4" col="70">QPADEV001C</span>
```

After turning this option on, the same section of the page would by default be generated as:

```
<span id="C0248" class=" nrm " row="2" col="48"> <%=screen(2,48,19)%></span>  <span
id="C0270" class=" nrm " row="2" col="70"><%=screen(2,70,8)%></span>
<span id="C0348" class=" nrm " row="3" col="48"> <%=screen(3,48,19)%></span>  <span
id="C0370" class=" nrm " row="3" col="70"><%=screen(3,70,10)%></span>
<span id="C0448" class=" nrm " row="4" col="48"> <%=screen(4,48,19)%></span>  <span
id="C0470" class=" nrm " row="4" col="70"><%=screen(4,70,10)%></span>
```

Notice that all text constants (or apparent constants...) are rendered as screen tags, e.g. `<%=screen(4,70,10)%>` instead of `QPADEV001C`.

To switch to this mode, open the Advanced Configuration options from the Tools menu and find the CustomPageReplaceSpans option. Set it to 1.

Considerations For Using Custom HTML

When you choose to enhance a page with custom HTML there are a number of considerations that should be part of that decision. The following sections discuss these.

Optimizing the Initial HTML

If you intend to use custom HTML to add features into your page you should get it as close as possible to your ultimate result using the Active Page customization tools (tables, windows, etc.) before clicking the Create button to move into editing the HTML manually.



If you haven't read it already, the section "Review the Basic HTML" on page 316 has an introductory discussion of this topic.

Consider this illustration:

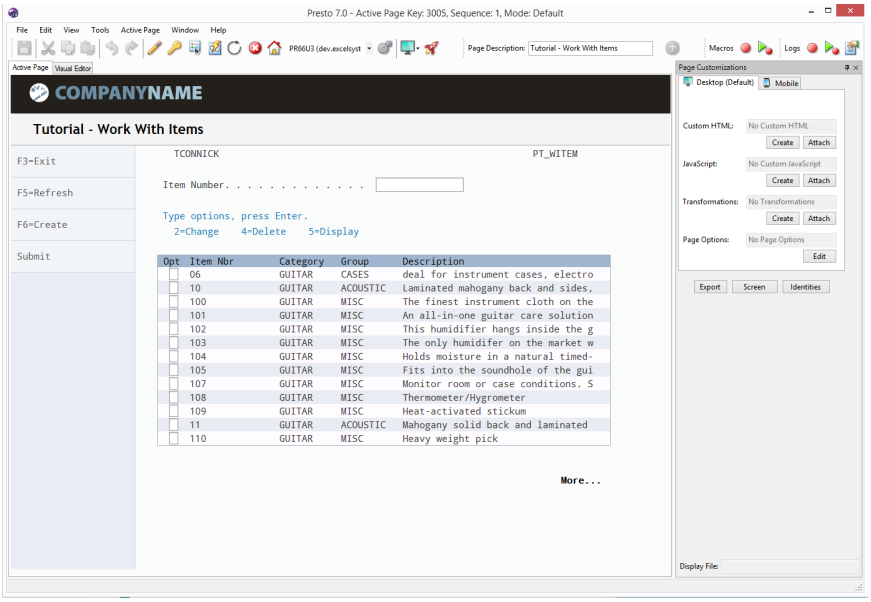


*Figure 2-4. Getting your tables and other features set up *before* further customization.*

Once you click the Create option to build the custom HTML, all record of your previous changes (tables, windows, etc.) is mapped into the generated HTML. Note that it is NOT saved anywhere else.



Presto doesn't keep track of multiple versions of your customizations as you prototype changes. This means you can't return to an interim level of changes unless you explicitly save off your HTML changes.

Removing Custom HTML

If you decide you no longer want to use the custom HTML you can unlink it from a particular green screen page or delete it entirely.

You can remove custom HTML from the Active Page options. To delete custom HTML entirely you must access it from the Manage Pages list. You can get to this list from the Presto Designer's Tools menu.

Applying Custom HTML to Other Pages

It is possible to both manually re-attach custom HTML to a page it was previously built over, or link existing custom HTML to a different green screen page.

You may need to do this if, for example, your green screen program has changed enough that Presto no longer recognizes it as the original screen. The symptom of this would be that after making some changes to the original green screen version of a program, when you call your program in the browser it loads an original, uncustomized version of the page. When this happens you can re-attach your custom HTML from the Manage Pages interface, taking care to make sure that any changes you've made to your original program are also reflected.

Managing Static Text vs. Output-Only Fields in Custom HTML

A key concept to understand as you modernize screens is the challenge Presto faces in distinguishing static content on your pages (e.g. text strings) from variable data values. Because Presto is only looking at the IBM i datastream there are limits to what the software can tell with certainty about that data. So when you first create custom HTML you may find that Presto has recognized some fields as dynamic values that may change each time you run the program (and generated <screen> tags for them), while for other fields, it has rendered their content in the HTML verbatim, as static text.

In the next sections we'll explore some of the ramifications of this.

Using Tables to Flag Subfiles as Dynamic Text

Presto treats the first text element in a table cell as variable. The second or subsequent text strings in table cells, and text strings outside table cells, are treated like text constants. Therefore, on screens with many fields (such as subfiles), it is a good practice to build the best possible table over your variable data before creating custom HTML.

As an example, recall this page we looked at before:

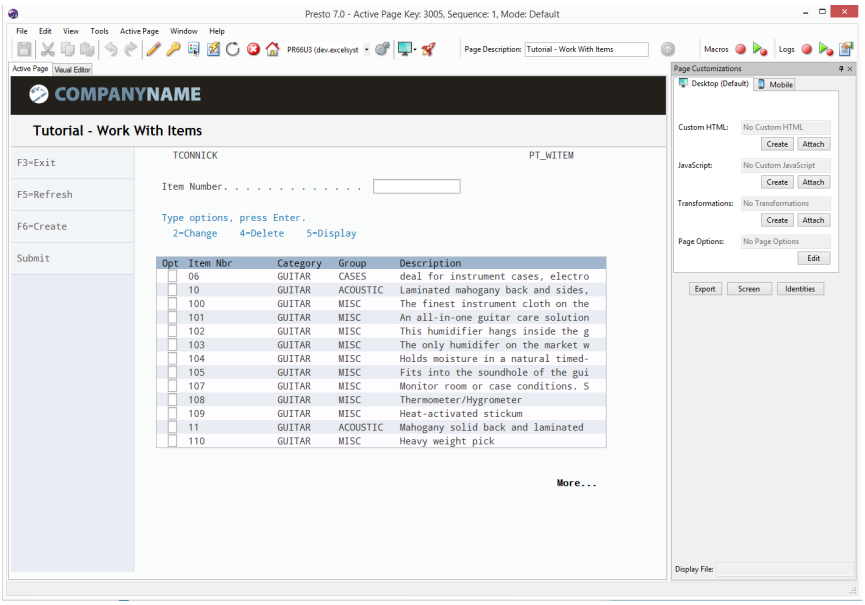


Figure 2-5. A list of items within an HTML table.

If we create custom HTML over this screen without having built a table over the subfile, a typical row in the HTML looks something like this:

```
<%=field(9,3,1)%>
<span id="C0905" class="<%=attr(9,5)%>" row="9" col="5"></span>
<span id="C0906" class="<%=attr(9,6)%>" row="9" col="6">06     </span>
<span id="C0922" class="<%=attr(9,22)%>" row="9" col="22">GUITAR
</span>
<span id="C0933" class="<%=attr(9,33)%>" row="9" col="33">CASES
</span>
<span id="C0944" class="<%=attr(9,44)%>" row="9" col="44">A great deal for instrument
cases</span>
```

[We have adjusted the line breaks to make this more readable.]

In this HTML notice the following:

1. On the very first line above the Options field is correctly identified as a variable (i.e. a field, not static text) and Presto has inserted an input field for it: <%=field (9,3,1)%>.
2. However, if you look through the rest of the content, you can see Presto is interpreting the values in the subfile as static text. For instance, on the third line of the HTML shown above you can see a static entry for the item number, `06`, the first item in the subfile list.

The problem here is this:

- Now that we have created custom HTML over this screen, the preliminary HTML generated by Presto for this custom HTML will be used as a template to be applied to this screen every time it's accessed in a browser.
- The next time anyone runs this program, or pages up & down through the list, Presto will check its database for a matching custom HTML and find this one.
- But because this page has been saved with static values in the list, in Presto they will always see the same data!

Clearly, this would be confusing.

To address this, your custom HTML needs to use a dynamic screen tag for any piece of text that can vary in the original green screen program. In subfiles, the best way to handle this is to build a table over the list BEFORE creating the custom HTML.

So in a case like ours above, we would remove the custom HTML and go through the steps to add a table to this page to give us a better starting point to further customization (or use the subfile table that Presto will automatically add to the screen, using the default settings).

Adding Green Screen Content to Your HTML

In the section above we introduced the concept of how Presto uses replacement values in your HTML to map in variable content from your green screen panels. We also focused on ensuring that we would have the best possible starting point in our custom HTML by creating a custom table over our data.

However, there are often other variable data elements on your screens that aren't necessarily suited to being in a table, or that lie outside an existing table you've already built over a different area of the screen.

Here is an example of an Order Entry listing, where we have built a table over the subfile data, but there is also some variable data (coming in from the calling program) in the header, the Customer Name:

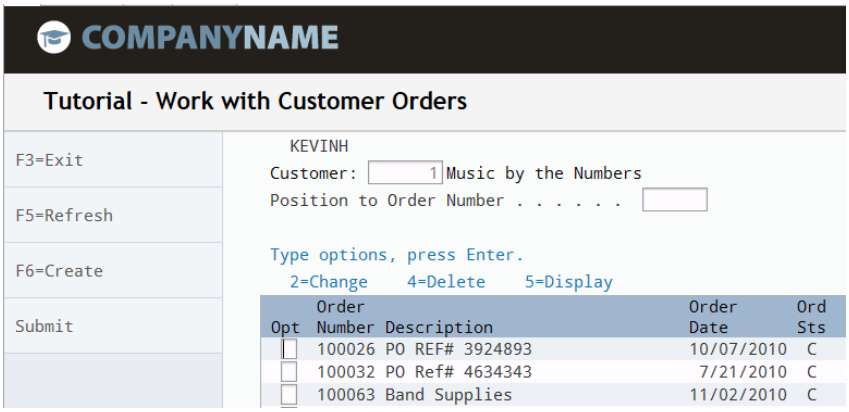


Figure 2-6. A list panel, with some variable data in the header, outside the table.

Because this field isn't in a table cell, nor input-capable, it isn't possible for Presto to automatically determine it's a field (and dynamic) rather than static text.

When we create custom HTML over this list, the top of the HTML includes a section that looks like this:

```
<span class="<%=attr(2,20)%>" id="C0220" style="  " col="20" row="2">Music by the
Numbers                                  </span>
```

Notice that the Customer Name is rendered as a static, text value: `Music by the Numbers`.



Key Concept. To reiterate - the practical implication of this is that if you leave this section of HTML this way in your custom HTML, every time your users come to this screen for any customer, they will see the same value (in the Presto version of the page, of course; the green screen is unaffected).

Because it isn't always possible for Presto to tell by reading the datastream whether in the originating program a text string is a field or plain text, in some cases you need to edit your HTML afterwards to correct this.

To do this, you just delete the static text in the HTML area, and replace it with a <screen> tag. You can type in the <screen> tag by hand or use the Presto Designer to drag & drop a string from your green screen display below the custom HTML.

So following on the example above, in the illustration below we have deleted the static Customer Number, and are in the process of dragging the value highlighted in the section at the bottom and dropping into the appropriate place in the HTML above:

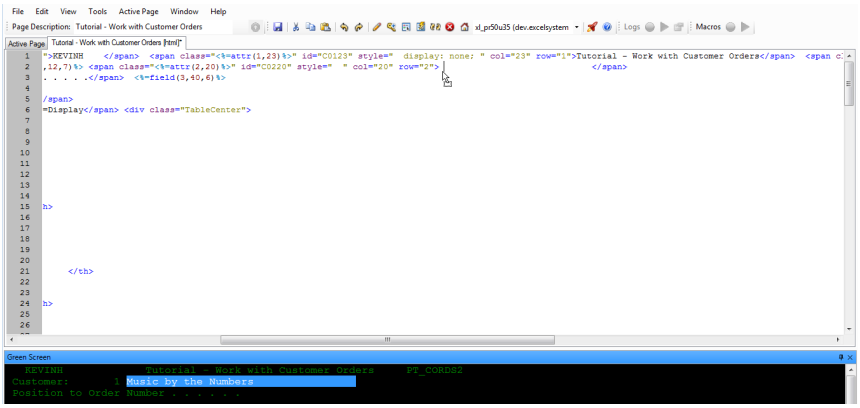


Figure 2-7. Dragging an area from the original screen, to the HTML rendering of it.

We are about to replace the static value of 'Music by the Numbers' with a `<%=screen(2,20,36)%>` tag. This tells the program that whenever that page is loaded it needs to display whatever is in those positions of the original green screen program (row 2, column 20, for 36 positions).



If you have trouble dragging & dropping to the right spot (because the code is so wide, and line lengths vary widely), drag the value to any spot in the HTML window, then cut and paste it from there to the target location.

You can also just type the screen tag by hand.

In the next chapter we will look at more sophisticated features you can add to your pages using a combination of your fields and JavaScript.

Using Scripts in Custom HTML to Retrieve Screen Values

Please refer to the section "Support for Retrieving Dynamic Values from the Screen" on page 299 for an example of code you can use to retrieve values from the screen to use in scripts in your custom HTML.

CHAPTER 15

Chapter 15 - Working with Your Presto Skins

For basic instructions on working with your skins and creating new ones please refer to the online Presto Tutorials. In the online tutorial, Creating a Custom Presto Skin, guides you through the steps to get started on this process. This chapter covers a few specific topics related to the skins, but if you have questions about your own customizations needs, please contact Technical Support!

About Presto Skins

Presto renders your green screen pages in a browser using a combination of HTML, JavaScript, page templates and CSS files.

A familiarity with your skins is important from both cosmetic and functional points of view. The HTML and CSS components of your skins control many aspects of the look and feel of your pages and you will want to work with them as you create a new page layout that matches your organization's design standards or requirements. However, there are also a number of useful functional enhancements that more advanced users can make via the HTML, CSS and JavaScript files that are shared by all the pages on your site. By leveraging some of the features we have built into the files shared by all your pages, you may save yourself from having to customize individual pages. Please refer to the chapters on using custom scripts for more details.

Skin Support for Mobile Devices

As the use of various types of mobile devices has exploded it has become increasingly important to design your interfaces to be readily accessible to that audience. Your Presto skins include a number of features related to supporting mobile applications, including:

- Custom layouts optimized for mobile devices.
- A framework that detects the device type and optimizes the layout accordingly.
- Displays automatically resize when rotated.
- Support for gestures on iPad.
- A custom mobile keyboard with required 5250 keys.

You shouldn't have to do anything extra to benefit from these changes - just test your applications on the devices you're targeting. If you need assistance with further customizing your skins or screens for particular devices, please call Technical Support.

Examples

Here is an illustration of how a mobile device screen looks using Presto's out-of-the-box settings:

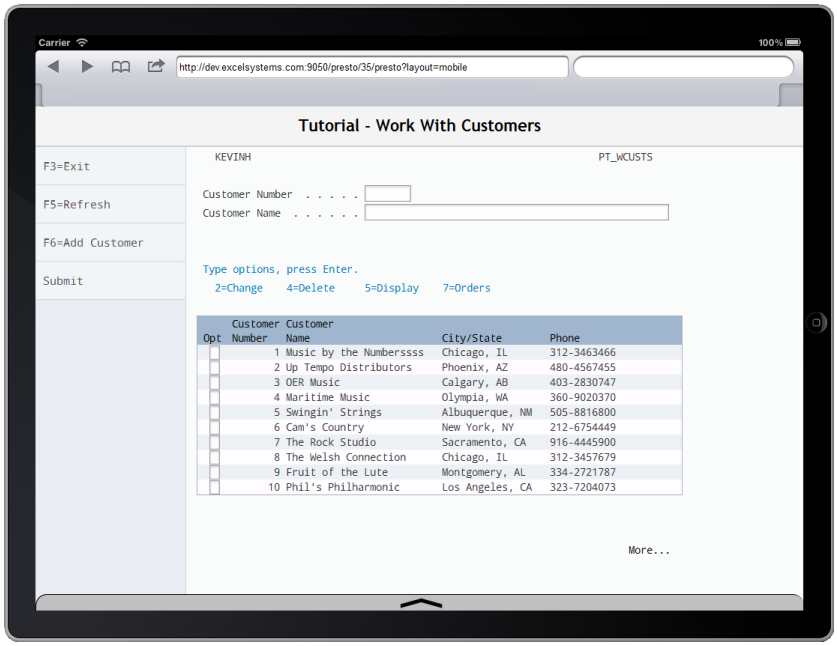


Figure 2-1. Sample subfile viewed on a mobile device.

In this illustration you can also see the handle for the extra keyboard you can pull up to use your function keys:

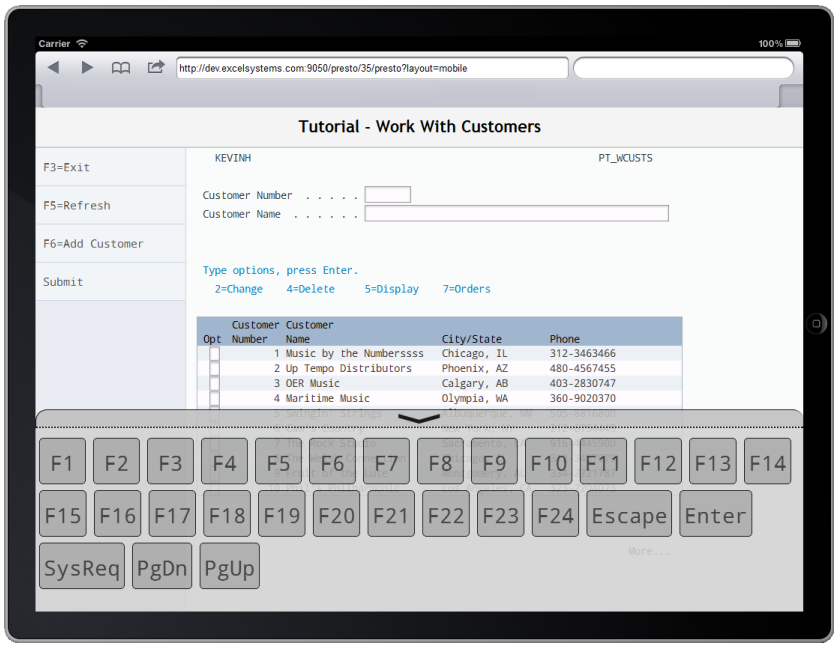


Figure 2-2. Mobile screen displaying extra IBM i keys.

You can test the skins with your own applications and devices.

The Presto IDE allows you to switch to this mobile view of your site. Select Mobile from the button in the Presto toolbar:

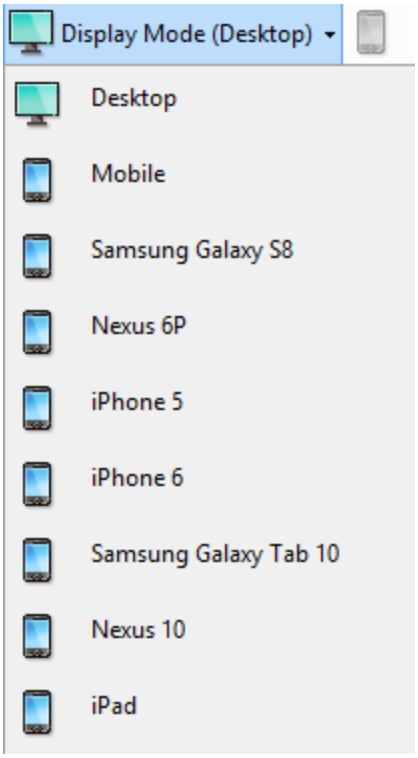


Figure 2-3. Displaying Presto display modes

Selecting Mobile loads the layout optimized for Mobile Devices.

Using jQuery Mobile

To make it easier to customize your green screens for tablets and mobile devices Presto includes jQuery mobile in the IFS files installed with Presto by default.

If you need some guidance on getting started with modernizing your green screen application to run on tables and mobile devices, contact Fresche Technical Support for a quick overview, or Fresche Professional Services.

CHAPTER 16

Chapter 16 - Using Presto with RPG OA

RPG Open Access (OA) is a technology developed by IBM, included with V6R1 at no additional cost, which you can use with Presto to give your RPG programs a web GUI. RPG OA was designed as a development strategy to allow organizations to reuse existing skills when creating modern RPG-based applications. It does this by providing a way to use the simple and well-understood RPG I/O model to access resources and devices not natively supported by RPG[1].

RPG OA complements other Presto features with these notable benefits:

- Presto can immediately recognize a subfile and create a scrollable jQuery grid, with sortable column headings, of the subfile's entire valid data set.
- You can seamlessly combine OA screens with other pages that you have web-enabled using Presto's 5250 conversion and design tools.
- Presto can recognize field types in OA programs, to which we can automatically attach unique CSS classes. For example, the handler program attaches datepickers to all date fields.
- Presto uses fields names to assign a class ID and label to all fields. This helps you identify fields for custom features.
- When you use Presto as part of your modernization strategy you can use the powerful Presto Visual Editor to build your pages with images, dropdown boxes, auto completes and much more!

RPG OA is particularly well suited for new development of applications you don't intend to present via 5250 (though you can also use it as part of your toolkit to modernize your 5250 applications).

How it Works

Consider this simplified illustration of how Presto works in a 5250 environment:

[1] http://www-01.ibm.com/common/ssi/rep_ca/4/897/ENUS210-114/ENUS210-114.PDF

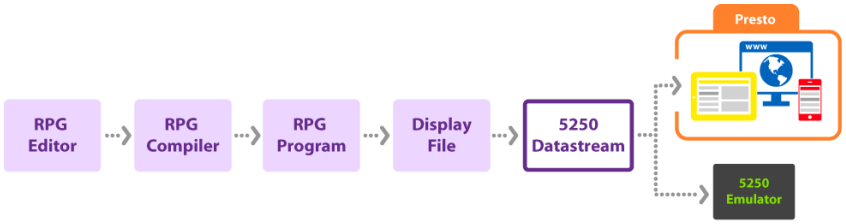


Figure 2-1. Simplified schematic of Presto in a 5250 environment.

Here you can see the various steps that together generate a 5250 datastream that you can either deliver to a terminal or render as a web application with Presto.

In an OA development model, you may not build your program to deliver a 5250 datastream at all, and instead use a handler program to convert your output:



Figure 2-2. Presto OA schematic.

When you are using OA you still have an RPG program with fields defined in DDS, but it is not designed or built to ever be used with a 5250 session. When you add the Presto handler keyword to the F spec of your program, that makes it interact with the Presto OA Handler program instead of the 5250 datastream.

When you use Presto to deliver an OA program, you'll use the Presto Designer to customize the page layout and grid that your users will work with, rather than building a display file for your intended page layout.

Considerations

As a modernization option, there are also some considerations you should keep in mind when incorporating RPG OA programs:

- Because no screen constants are delivered to the handler program, you'll build screen text using the Presto Visual Editor.

  As noted previously, for this reason and others, OA is particularly suitable for new development where you have not invested any effort yet in screen design and are building the display specifically for delivering the program via a browser.
- You can also add field formatting information (color/bold/position/etc.) via the Presto Visual Editor or, in the case of fields in the grid, through JavaScript.
- If you are converting existing programs to OA and using the Presto handler, they can no longer run via 5250 unless you have an alternate compile method that only conditionally includes the handler.
- Programs must be RPG IV.
- Minimum system requirement is V6R1.

The following sections describe the technical requirements and how to use RPG OA and Presto to deliver web applications.



Presto does not currently support OA and when you are using Webfacing. If you need to use these two technologies together, please contact Technical Support to press your case!

Important Performance Consideration for Upgraders

Presto 5.6 introduces a change to how your programs interact with the server component of the software. Because this is a far-reaching change (e.g. there is some risk), existing installations will continue to use the original approach until you explicitly switch over.

To switch to the new method to go Tools>Advanced Configuration to open your config.bws file and find this line:

```
<ParametersVersion>1</ParametersVersion>
```

Change this setting from 1 to 2 and save your changes.

When you use version 2, the Presto program running in the browser will only return modified values back to the server, which in our testing has produced significant performance improvements, particularly with OA programs.

RPG Program Requirements

RPG OA is designed to deliver the program output through a handler program specified in your program's F specs. In the Keywords section of the F specs of your `WORKSTN` definition, add the following code, referencing the Presto OA handler:

```
HANDLER('PR_OAHDLR(EP1)')
```

You can optionally include a library reference to `XL_PRESTO` or the name of our current Presto library.



Regardless of whether you include a library reference in your programs, you must have the handler library in your end users' run time library list.



Note that if you use SEU to make the code change above you will get a keyword not recognized error. However, you can still exit and save your change.

The handler program generates a list of fields on the screen and builds a jQuery grid using the field names as column headers.

Here is some sample code from a very simple program that illustrates the inclusion of the required F spec:

```
* Writes a subfile to the screen. Enter ends.
```

```
FOASUBFILESCF E WORKSTN Handler('PR_OAHDLR(EP1)')
F SFILE(R_SFLRCD:SflIdx)
FMU_ITMF IF E K DISK
D SflIdx S 4P 0
/free
SETLL (*LOVAL) R_ITMF;
DoW Not %eof;
READ R_ITMF;
If Not %eof;
SflIdx = SflIdx + 1;
WRITE R_SFLRCD;
EndIf;
EndDo;
*In51 = *ON;
ExFmt R_SFLCTL;
*InLR = *ON;
Return;
/end-free
```

Customizing RPG OA Programs with Presto

RPG OA is pushes the data to the browser without the default layout content that you may be accustomed to seeing as a starting point in Presto when modernizing existing 5250 applications. From the starting point delivered by the handler program, you can use the Presto Designer to add your screen text constants and adjust field locations and characteristics.

Here is an illustration of the program generated from the code sample above, as it first appears in Presto when we call it from a command line:

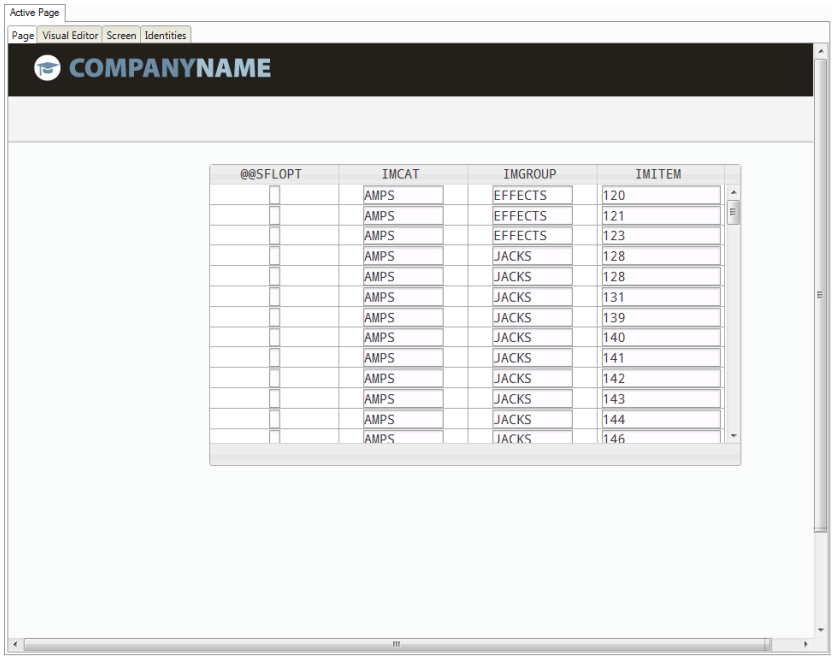


Figure 2-3. OA program as first delivered to the browser.

In this illustration you can see that the program is recognized as a subfile and all the records are loaded in the grid. Field names are used as placeholders for the column descriptions in the grid. This particular program doesn't happen to have any additional fields outside the subfile.

Starting from an initial layout similar to the one shown above you can add further features through the Presto visual editor or page JavaScript.

Here is another illustration of this program after making some custom changes:

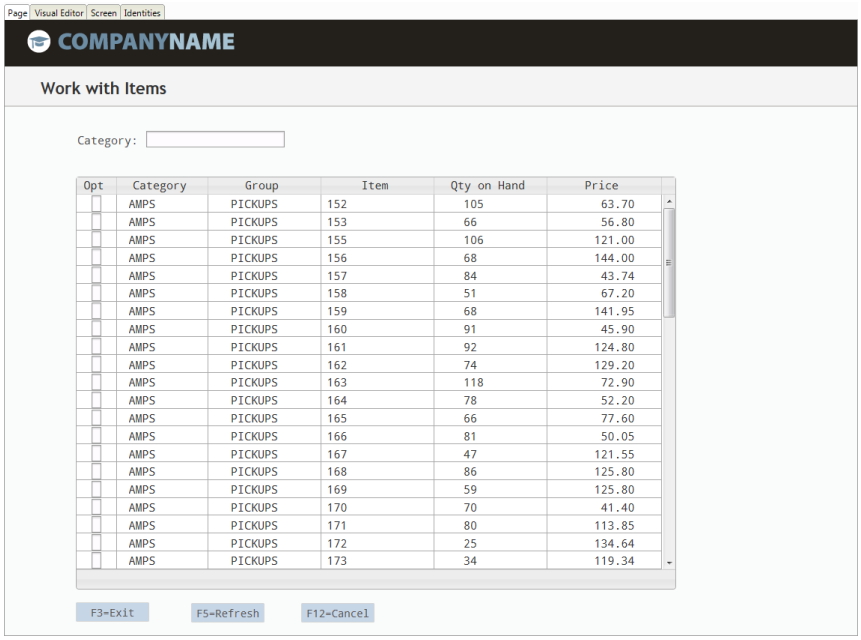


Figure 2-4. Changed OA program.

In this program you can see that we have added a field and text to the header, as well as function keys below the list.



Note that all OA screens will initially have a page description of 'OA Page'. As you customize your pages, it is important that you give each page a meaningful, unique name for future reference.

Adding Content via Transformations

The transformations include a few labels and buttons:

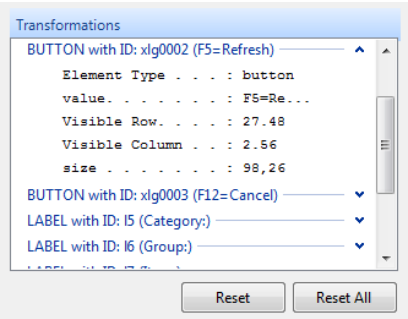


Figure 2-5. OA program transformations.

You can find instructions and examples of buttons earlier in the guide, or you can get help from Technical Support. You can also use JavaScript and jQuery to change grid characteristics.

About the jqGrid jQuery Grid Plugin

jQuery scrollable grids support many powerful features and you can find myriad online articles and tutorials on how to customize grids. You can easily 'skin' your grid, as well as change table attributes, column orders, data formats and sort orders.

And check the Fresche Knowledge Base for ideas and code samples!

Modifying Grid Properties

Here are lists of the properties which can be modified:

- grid - http://www.trirand.com/jqgridwiki/doku.php?id=wiki:options
- columns - http://www.trirand.com/jqgridwiki/doku.php?id=wiki:colmodel_options

Contact Technical Support if you have questions!

Making Your Subfile Grids Sortable

To enable column sorting for Presto OA subfile grids, add the following line your page's custom JavaScript:

```
Presto.bAllowOaSort = true;
```

This will turn on sorting if you click the column headings for that page.

If you wanted all grids to automatically be sortable, add this line to your global JavaScript file you could put that into your global custom javascript as well.

The Presto OA Subfile Grid Event

For OA programs which write a subfile to the screen Presto triggers an event before the grid (subfile) is loaded: BeforeLoadGrid.

You can use this event to modify the grid properties, override column headers, and hide fields. You can also use it to read the subfile data before it is loaded and apply any conditional actions.

The Presto OA Grid API

This is the technical documentation for the Presto OA Grid API.

Presto.Oa.Grid.GetField('FieldName')

This method takes a string representing the field name you want to modify in the grid. It will return a field object which has two methods currently available.

Presto.Oa.Grid.GetField('FieldName').Set('Option', 'Value')

This method will set the specified option to the given value on the field object. This method returns the Field object, so it is possible to chain calls to it.

These are the available options:

Option	Description
Align	String. Accepted values: 'right', 'left', 'center'.
Hidden	Boolean. Accepted values: true, false.
Input	Boolean. Accepted values: true, false.
Sortable	Boolean. Accepted values: true, false.
Width	Integer. Accepted values are any standard integer.

Presto.Oa.Grid.GetField('FieldName').SetOptions({Option: 'value'})

This method sets the specified options to the given values. It requires an object consisting of option/value pairs.

The available options are the same as Set (above).

Presto.Oa.Grid.SetSize(width, [height])

This method will set the overall grid size. The height parameter is optional, and if not specified the default height will be used.

Presto.Oa.Grid.SetSortable(bool)

This method will set the entire grid to be sortable if passed true, or only specific columns (Defined using GetField.Set/SetOptions) if passed false.

Presto.SetScreenTitle('Title')

This method will set the text that is added to the <h1> title element. Currently Presto utilizes a specific location on the screen to get the title, but as OA programs don't have a "screen" there is no information it can automatically grab.

Example

Here is a code sample:

```
jQuery(document).one('BeforeLoadGrid', beforeLoadGridHandler);
function beforeLoadGridHandler(event, oaJson, gridProperties)
{
        // Samples of setting grid field properties
        Presto.Oa.Grid.GetField('IMCAT').Set('Input', false);
        Presto.Oa.Grid.GetField('IMGROUP').SetOptions({
                'Input': false,
                'Title': 'Item Group'
        });
        Presto.Oa.Grid.GetField('IMITEM').SetOptions({
                'Input': false,
                'Hidden': true
        });
        Presto.Oa.Grid.GetField('IMQTONHD').SetOptions({
                Input: false,
                Align: 'right',
                Width: 100,
                Sortable: true
        });
        Presto.Oa.Grid.GetField('IMRETPRC')
                .Set('Input', false)
                .Set('Align', 'right');

        // Example of setting the grids overall width
        Presto.Oa.Grid.SetSize(800);

        // Example of setting the page title
        Presto.SetScreenTitle('Work With Customers');
}
```

If you have any questions or feedback we would love to hear from you!

Demo OA Program

We ship a demo OA program named `OAITEMSFL` in `XL_PRDEMO`. This OA program is over the file `MU_ITMF` and has a basic subfile and subfile `option 2=Change`. Minimum OS for this program is V6R1.

CHAPTER 17

Chapter 17 - The Presto Skin Editor

About the Skin Editor

An important Presto feature is the ability to customize your site's skin.

You can use the Presto Skin Editor to select from preset color themes as well as quickly customize the appearance of function keys, headers, and subfiles. Additionally, you can easily update the your site's logo.

This skin editor applies these customizations to your entire site without requiring to know the HTML or CSS.

Note the following requirements to use this feature:

- Currently the skin editor only supports skins cloned from the base Nightfall skin.
- Skins cloned in versions of Presto before 6.5 are not compatible with the new skin editor. This is because the skin editor depends on recent changes made to the CSS layout.
- We recommend that you clone the Nightfall skin before changing it in the skin editor. The base Nightfall skin may be overwritten during upgrades.

You can find further information on the skins supported by the editor in our online Knowledge Base article 15134:

https://www.myfreschesolutions.com/kb?kbnum=15134

Using the Skin Editor

You can open the Skin Editor by going to the Tools>Settings menu option then clicking the Skins button:

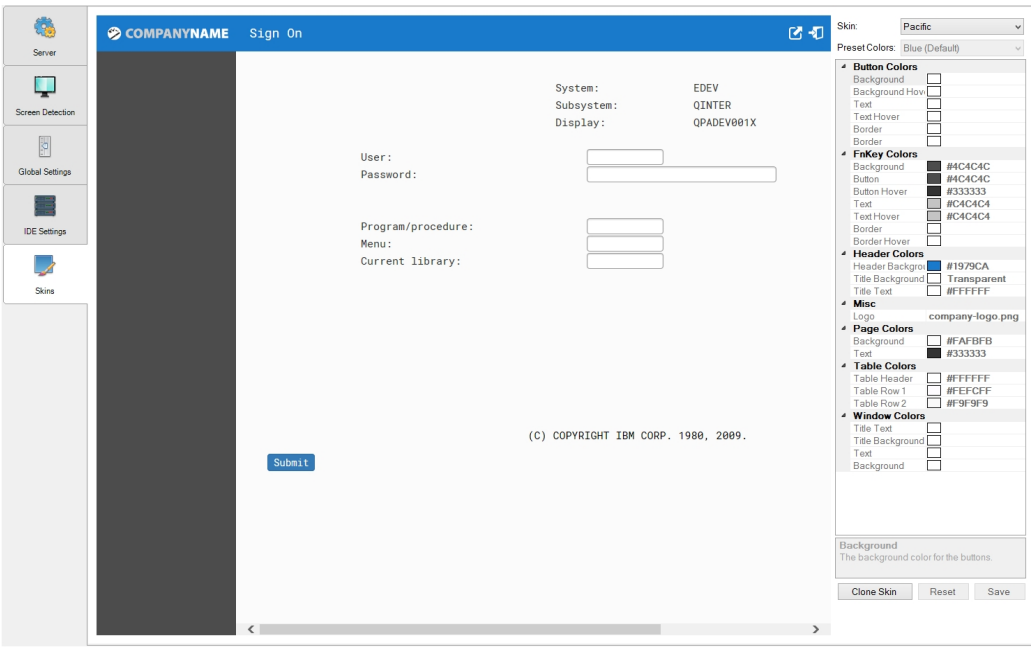


Figure 2-1. Accessing the Skin Editor.

To familiarize yourself with the skin editor, here are steps to change the color of the header background.

Start by clicking the area to the right of the Header Background property:

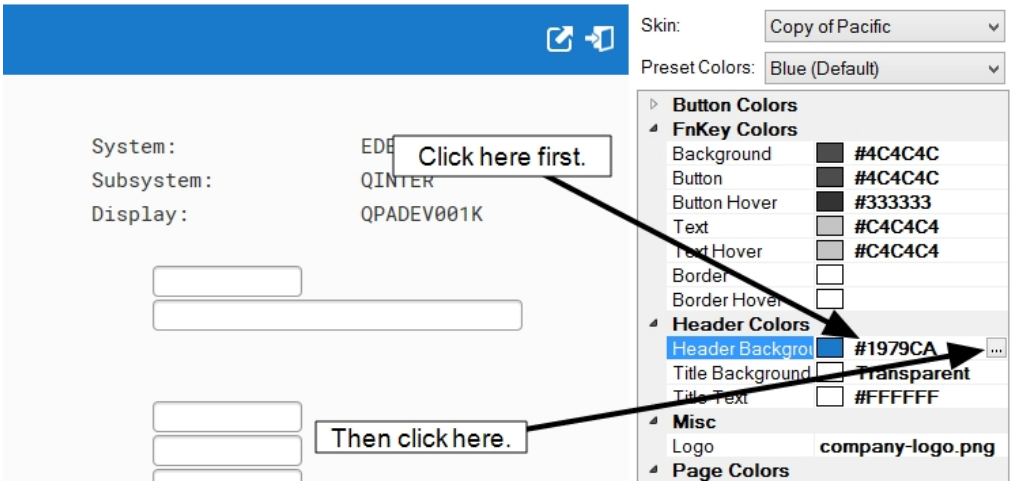


Figure 2-2. Changing the Header colors.

Then click the selection icon that appears (the box on the right, above).
When you click the icon, Presto presents this color picker:

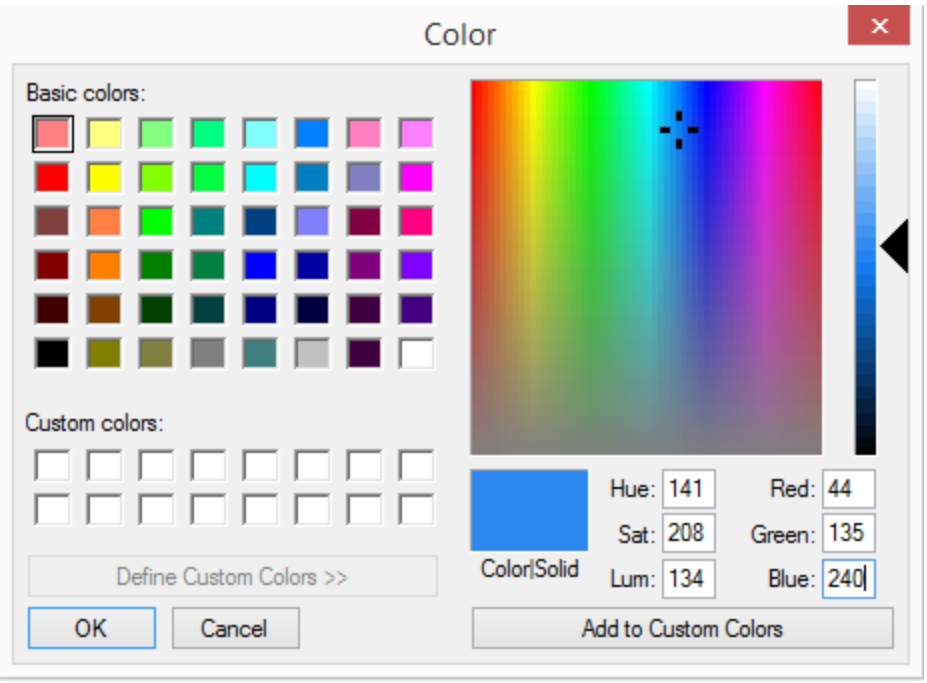


Figure 2-3. Selecting a new background color.

You can pick one of the basic colors on the left or define a color of your own choosing. Click OK to confirm your choice.

Notice the changes visible in the Skin Editor (listed below):

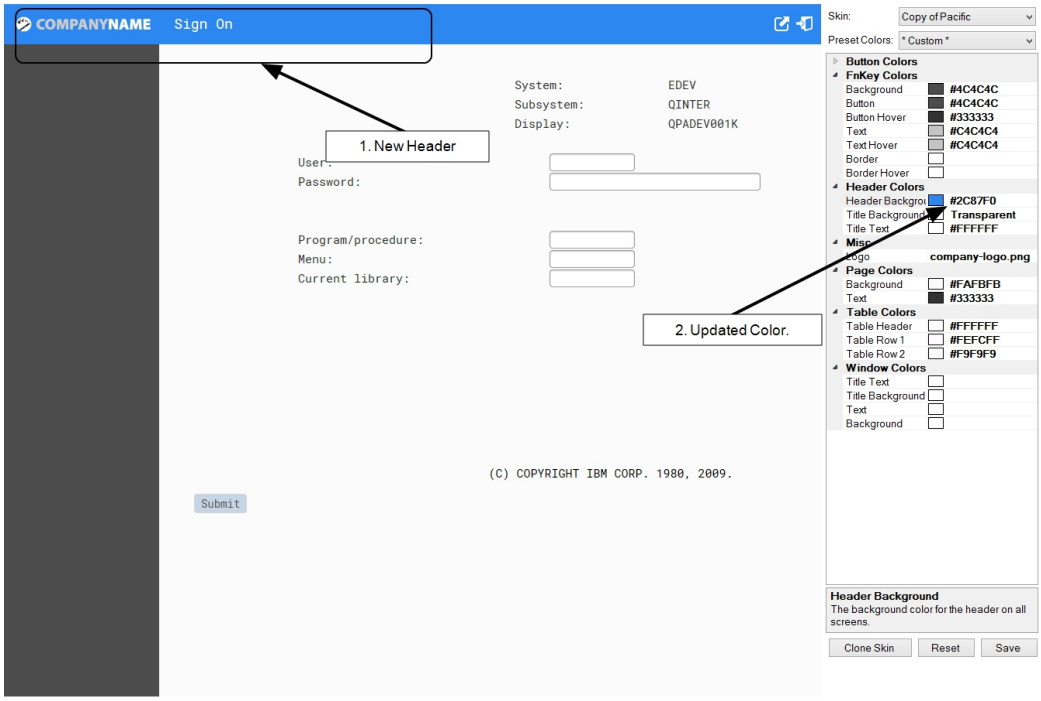


Figure 2-4. Reviewing the effects of making a change.

These are the things that happen when you change a property:

1. The Properties area shows both a preview of the selected color and color's hex value.

2. The change is reflected in the main window preview.

Using Preset Themes

Presto ships with four preset themes, which you can select in the skin editor. Themes include preset colors that let you quickly change the appearance of your current skin:

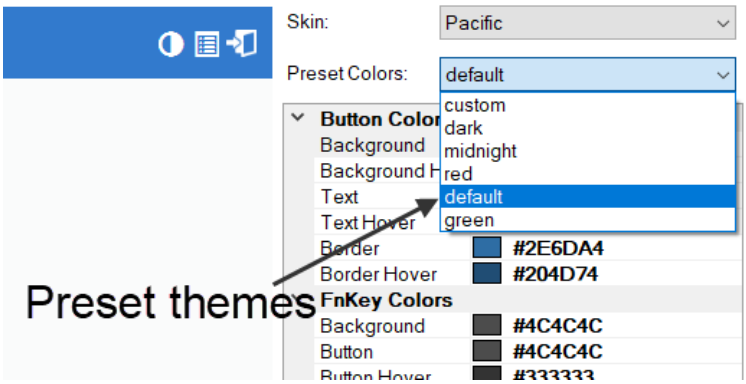


Figure 2-5. Selecting the Blue Preset

When you select the Red Preset, here is the result:

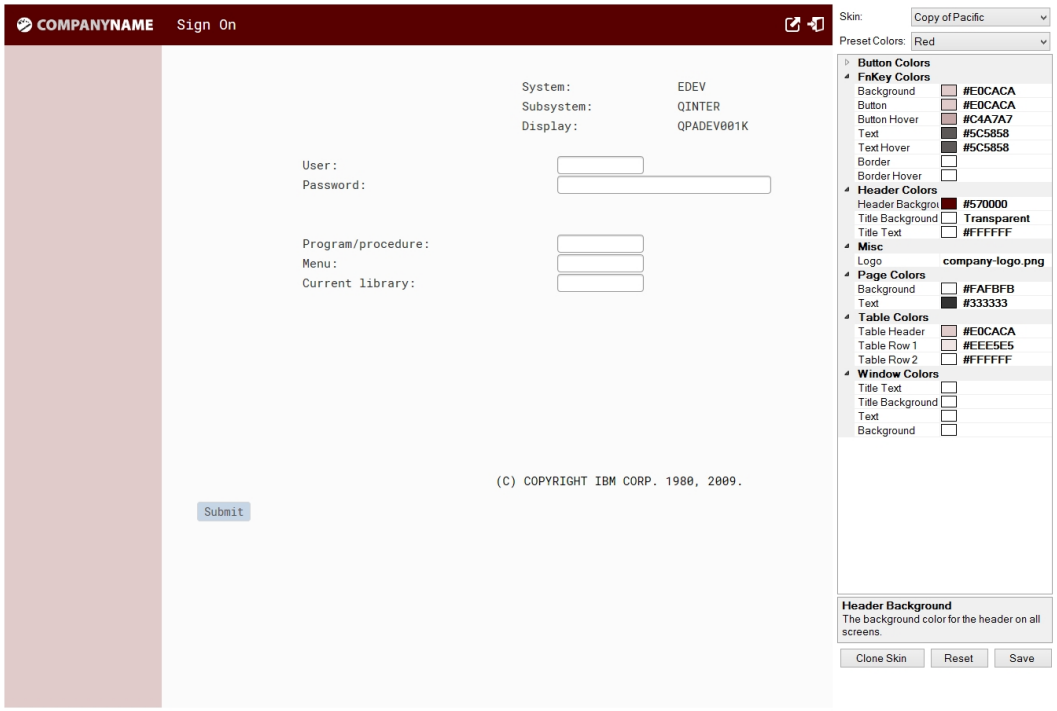


Figure 2-6. The Red Theme Presets.

Here you have the options to work with all the headers, pages, and table colors and logo in your site.

When you have customized the preset theme the way you like, Presto saves it as the “custom” theme preset.

Making Changes to Preset Themes

Preset Themes

These preset themes are included in your Presto IDE installation folder in a subfolder 'SkinThemes'. They are simple a collection of properties in a .txt file.

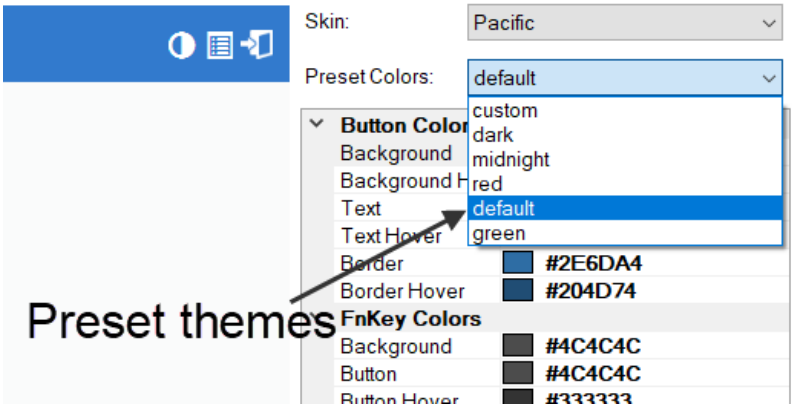


Figure 2-7. Reviewing the effects of making a change.

Here is an example of the preset default theme (additional properties are available in the actual file):

```
BGCOLOR=#FAFBFB
COLOR=#333333
FNKEYBGCOLOR=#CACAE0
FNKEYBTNCOLOR=#EEF7FF
FNKEYBTNHCOLOR=#5DB2FF
FNKEYTEXTCOLOR=#5C5858
FNKEYTEXTHCOLOR=#FFFFFF
HEADERBGCOLOR=#2C87F0
TABLEALTROW1=#EEF7FF
TABLEALTROW2=#FFFFFF
TABLEHEADBG=#D1D0CE
TITLEBGCOLOR=#5DB2FF
TITLETEXTCOLOR=#FAFBFB
```

Here are some things to note about the format of these .txt files:

- The filename is used to populate the dropdown value.
- The IDE only looks for .txt files.
- The file consists of several PLACEHOLDER=VALUE entries.
  - Each line corresponds to one placeholder entry.
  - Incorrectly formatted lines will be ignored.

The placeholder determines where the values should go in your skin's theme.css file.

Theme.css

The theme.css file found in the skin folder stores most of the color CSS for your Presto skin. The skin editor uses placeholders in the theme.css file in the format /*COLOR*/ where "COLOR" is the placeholder value.



If you plan in the future to use the skin editor, we recommend that you do not remove these comments for the placeholders.

You can open the theme.css file via Tools>Modify Skin>Theme.css

These are the available placeholder values:

Placeholder	Modifies
BGCOLOR	The background color for all screens.
COLOR	The color of general text for all screens.
FNKEYBGCOLOR	The background color for the function key sidebar.
FNKEYBTNCOLOR	The button color for the function key sidebar when hovering.
FNKEYTEXTCOLOR	The text color for the function key sidebar buttons.
FNKEYTEXTHCOLOR	The text color for the function key sidebar buttons when hovering.
FNKEYBORDERCOLOR	The border color for the function key sidebar buttons.
FFNKEYBORDERHCOLOR	The border color for the function key sidebar buttons when hovering.
HEADERBGCOLOR	The background color for the header on all screens.
TABLEHEADBG	The background color of table headers.
TABLEALTROW1	The background color of the first alternating table row.
TABLEALTROW2	The background color of the second alternating table row.
TITLEBGCOLOR	The background color for the title on all screens.
TITLETEXTCOLOR	The text color for the title on all screens.
FONTFAMILY	The font family on all screens.
BTNBGHCOLOR	The background for the buttons when hovering.
BTNBGCOLOR	The background color for buttons.
BTNTEXTCOLOR	The text color for buttons.
BTNTEXTHCOLOR	The text color for buttons when hovering.
BTNBORDERCOLOR	The border color for buttons.
BTNBORDERHCOLOR	The border color for buttons when hovering.

Placeholder	Modifies
LOGODISPLAY	Logo display CSS property.
LOGOVISIBILITY	Logo visibility.
LOGOALIGNMENT	Logo alignment.
LOGOHEIGHT	Logo Height in px.
WINTITLECOLOR	The text color of window titles.
WINTITLEBGCOLOR	The background color of window titles.
WINDOWBGCOLOR	The text color of windows.
WINDOWCOLOR	The background color of windows.

Changing the Skin Logo

The first change most customers want to make is the logo in the skin header:

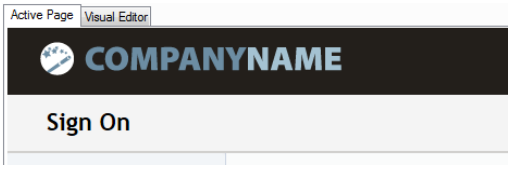


Figure 2-8. The current default logo in the Nightfall skin.

Step 1: Find or build a new logo.

For our example we Googled up a nice 'ACME' logo and saved it to our PC:



Figure 2-9. Our handy-dandy ACME logo.

After saving the image we changed the size, adjusted the transparency, and renamed it acme96x55.jpg.



For convenience, we incorporate the image dimensions into the logo name.

Step 2: Open the Skin Editor.

The next step is to change your skin to reference the new logo.

Select the option Tools>Skin Editor.

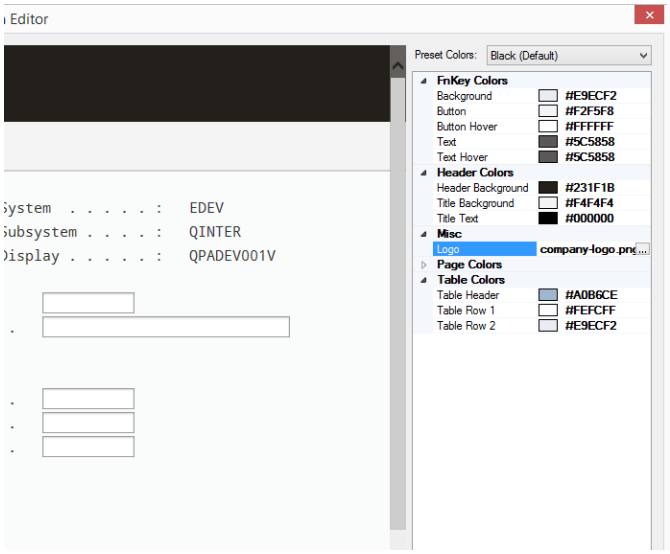


Figure 2-10. Selecting the logo in the skin editor.

When you select this option, Presto prompts you to select an image from your PC:

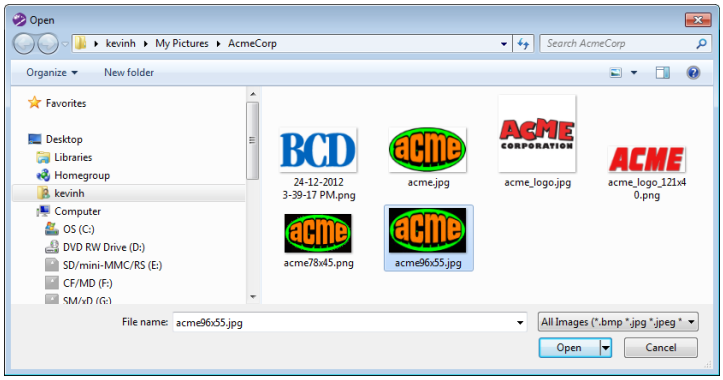


Figure 2-11. Selecting a logo for the skin.



The image you select is copied over the 'company-logo.png' file found in your environment's skin.

In this section we have changed the default image name from company-logo.png to acme96x55.jpg.

Note the following when changing the logo:

- The logo must be in .png format.
- Uploading a new logo overwrites the existing logo. That is, the new logo will overwrite any existing custom logo you have already uploaded.

After selecting this logo, Presto gives you a preview of your site with the new logo:

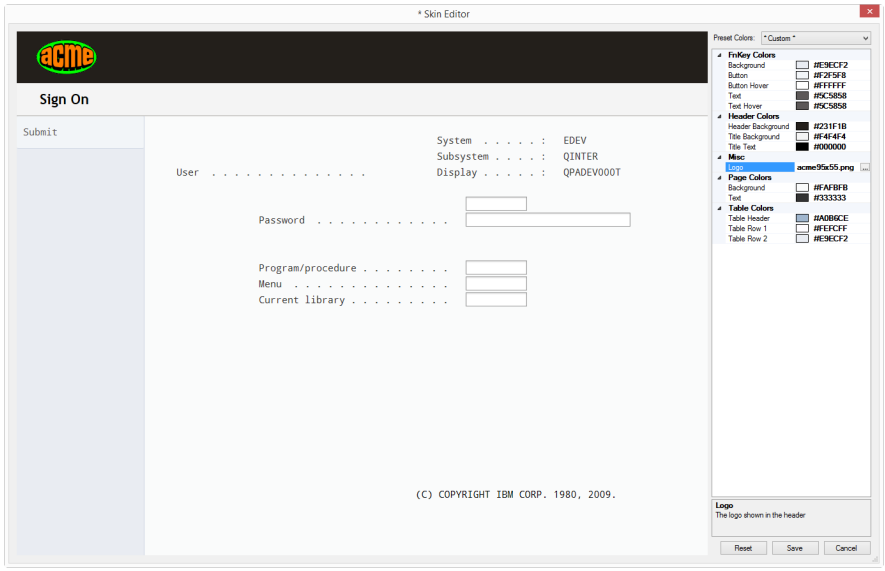


Figure 2-12. Preview of your site with the new logo for the skin.

If you are following these steps, set the reference to your own image and save your changes using the Save in the bottom left of the Presto Skin Editor.

Step 3: Test your page.

The easiest way to test your change is to launch your site (using the little rocket icon in the toolbar). This is the result (in our example):

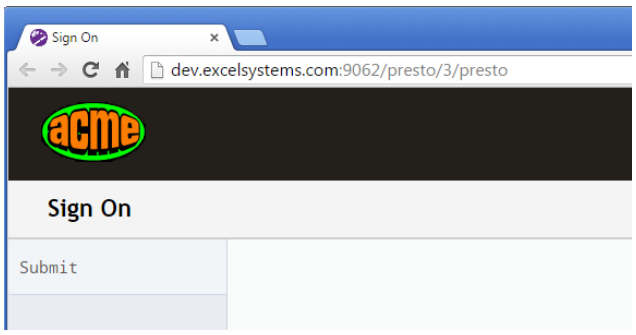


Figure 2-13. Changing the default logo in your Presto skin.

You can also reconnect to your Presto environment to reload all the files.

The change we've made here is about the simplest you can do. We kept the general appearance and structure of our Presto skin, and just switched the one image. The most important thing at this point is that you understand how the skins are built, and should you want to adjust them, where you can start from.

Color Options in the Skin Editor

Version 8.0 adds the ability to edit window colors in the Skin Editor.

There are additional new color property options in the skin editor for Buttons and Function Key borders.

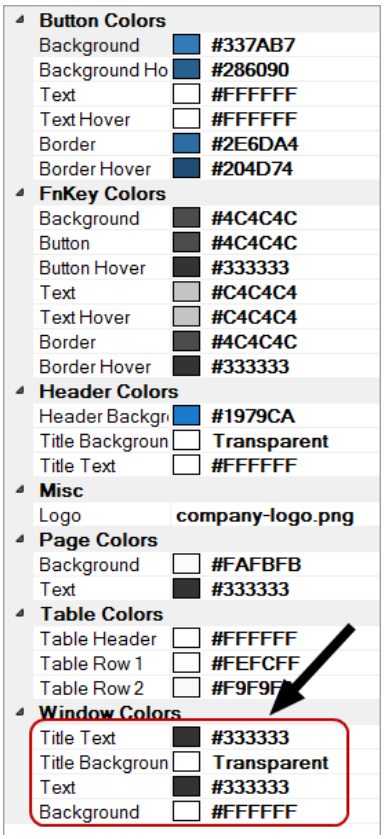
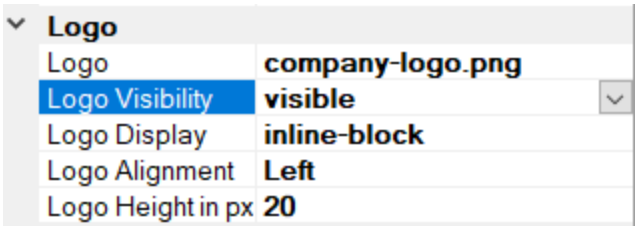


Figure 2-14. Skin Editor Color Properties

Logo Options in the Skin Editor

There are options for the logo visibility in the skin editor.



The logo visibility is a css property that allows you to control the logo space in the header. You can choose to display the logo as an inline-block element, or remove the element.

CHAPTER 18

Chapter 18 - Working with Mobile Customizations

About the Mobile Customizations

Presto simplifies the process of creating mobile friendly pages. You can use Presto's mobile mode to create and edit customizations for the mobile version of your site.

How Mobile Mode Works

The Presto designer has two layout views to display content, one for desktop (default) and one optimized for mobile - such as iPad, iPhone, and Android devices. When a user accesses a screen on a tablet or handheld device, Presto automatically detects the type of device and displays the mobile layout.

You can switch to the mobile layout in the Presto designer by using the controls in the toolbar.

Select the mode from this button in the Presto toolbar:

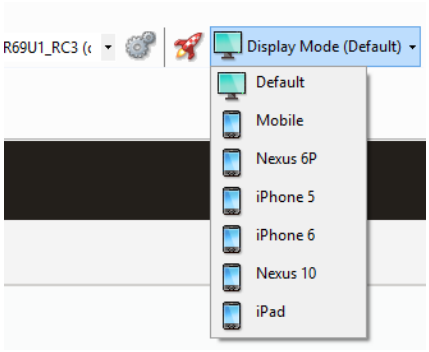


Figure 2-1. Switching from Desktop to Mobile Mode.

Select Desktop to view and edit the desktop view of your site and Mobile to view and edit the mobile site:

- **Desktop mode.** Presto loads the layout.html file, the HTML wrapper for the standard layout of your page.
- **Mobile mode.** Presto loads the layout_mobile.html file for the mobile layout of your page.

Here is an illustration of how a mobile mode looks in the Presto designer:

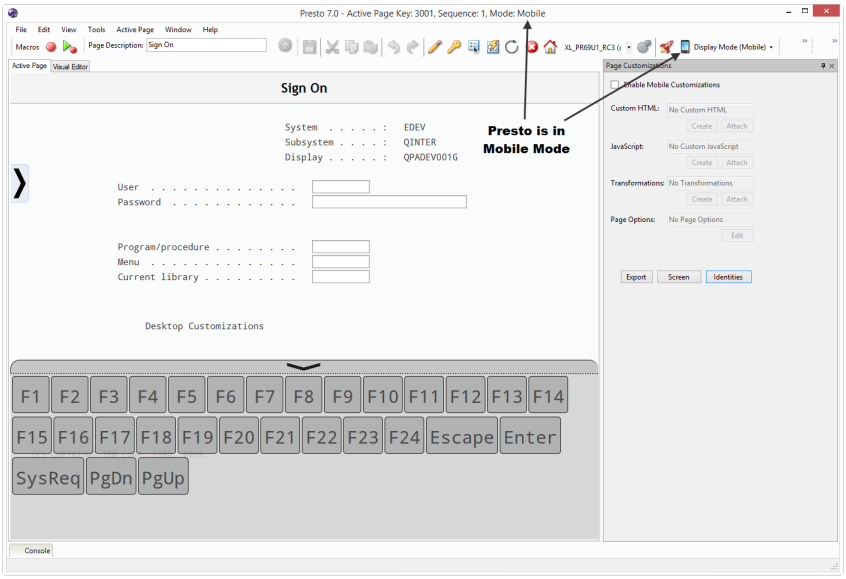


Figure 2-2. Previewing the active page in Mobile mode.

In this illustration above, you can see two things about Presto's out-of-the-box mobile layout.

- Hides the header and removes some images to optimize performance.
- Includes an extra mobile keyboard that you can pull up to use your function keys.

Enabling Mobile Customizations

You can use Mobile customizations to use the Visual Editor or edit the HTML and JavaScript to further enhance your screens for mobile devices. If you are customizing screens for mobile devices, the mobile cusomizations allow you to easily use the same Presto environment for both desktop and mobile layouts.

To edit mobile customizations, toggle to the Mobile mode in the Page Customizations area:

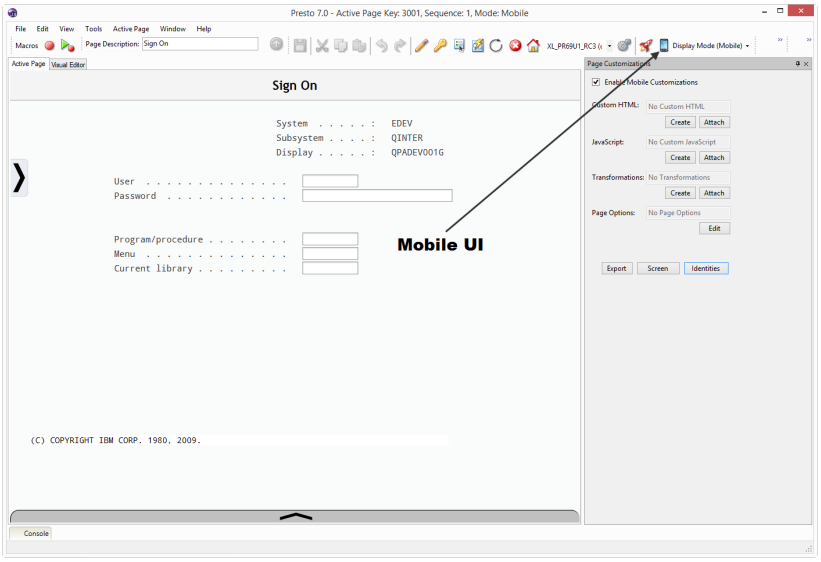


Figure 2-3. The Mobile UI.

The checkbox “Enable Mobile Customizations” in the Mobile tab is used to turn your mobile page customizations on and off:

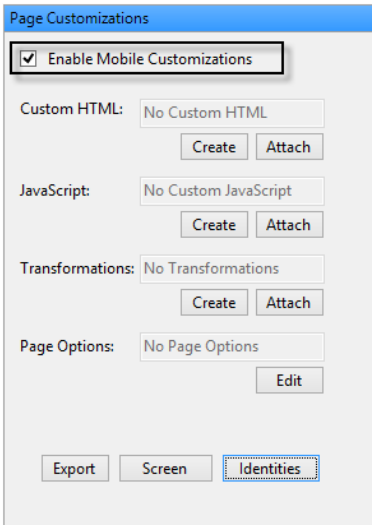


Figure 2-4. The Mobile Customizations option.

By default, your mobile page customizations are turned off. In the unchecked default OFF state, Presto shows the default desktop customizations.

Once you’ve decided to make mobile-specific customizations, check the box beside “Enable Mobile Customizations”:

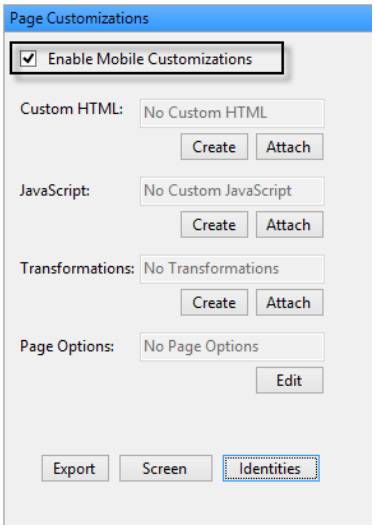


Figure 2-5. Enabling the Mobile Page Customizations.

Once you check this option, the mobile site is customized separately from the desktop site. In addition, your users will see the mobile customizations when they visit the page on a mobile device. This allows you to build dedicated customizations to support the mobile version of your site.

Note that toggling your mobile site on and off has no effect on the content. It simply determines whether to display your desktop (default) customizations or mobile

---

customizations to mobile devices.

Previewing Your Mobile-Specific Customizations

To go to the page with mobile customizations applied, click the rocket to launch a Presto session in your browser. This shows your page as it would appear in a mobile devices, with the parameter "?layout=mobile" appended to the Presto URL:

presto/presto?layout=mobile

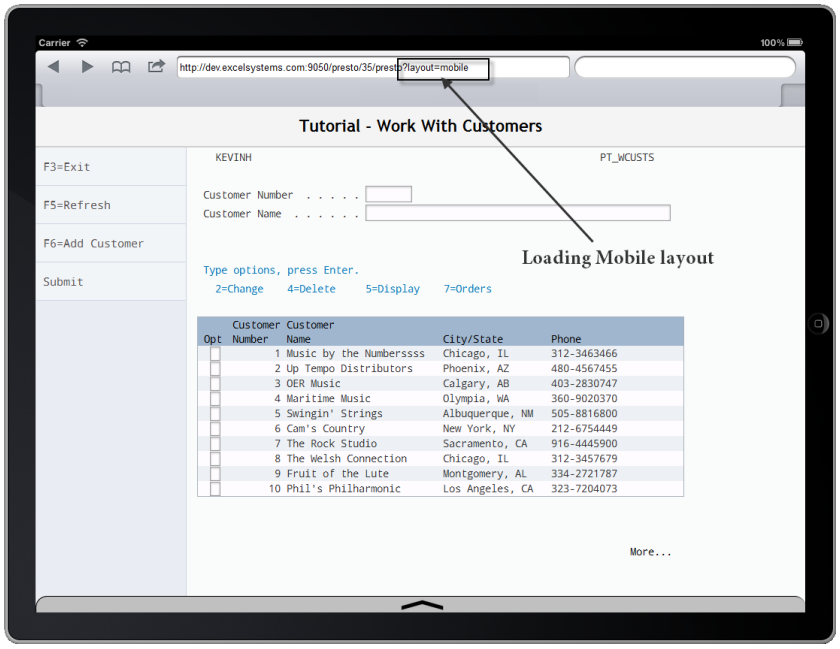


Figure 2-6. Viewing the Mobile Page Customizations.

Managing Mobile Customizations

You can work with the mobile customizations from the Manage Pages window.

To manage your custom changes, select the Pages option from the Tools menu. Here is an illustration of the window presented:

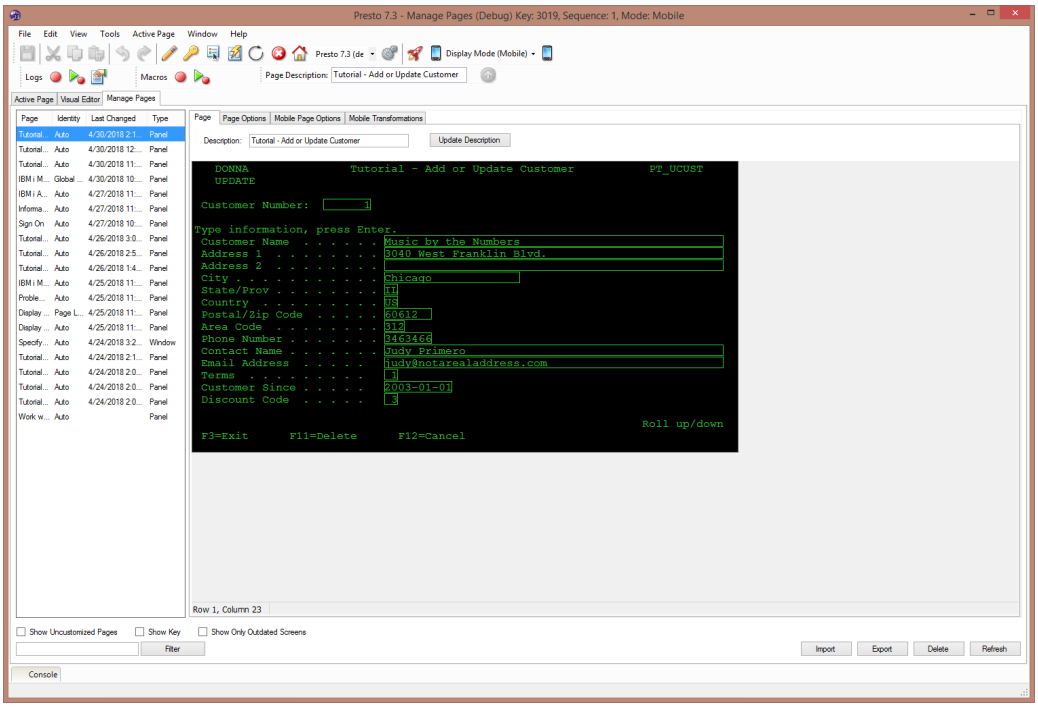


Figure 2-7. Presto's Manage Pages interface.

The Presto Manage Pages window (from the Tool menu) includes new tabs that list and allow you to work with your Mobile customizations.

You can also work with the mobile customizations, and from there see the pages to which they are linked. Here is an illustration of a mobile customization accessed from that tab:

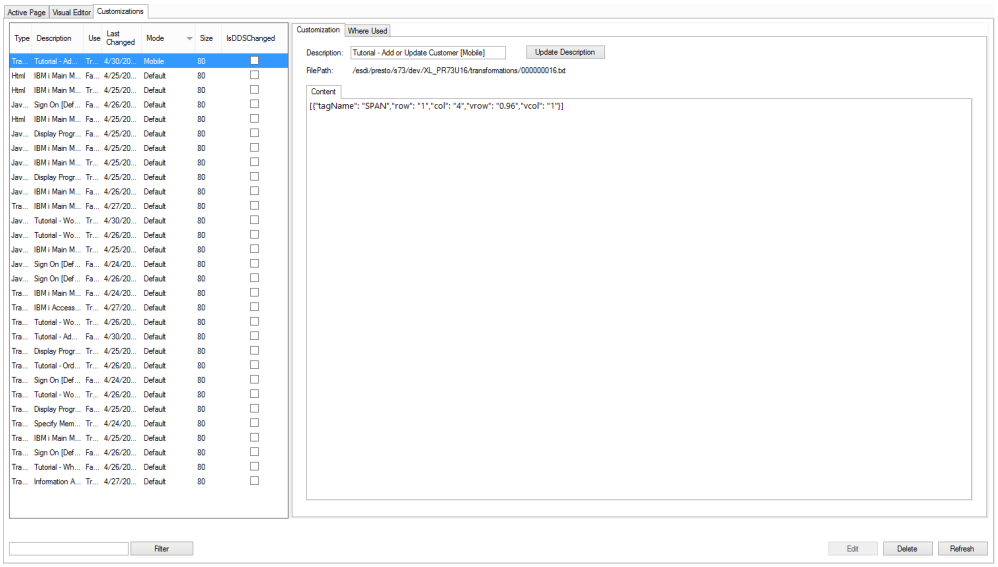


Figure 2-8. Viewing mobile customizations and what pages they are linked to.

The list on the left includes a column with the heading "Mode" that displays whether the customization applies to the Default or Mobile mode.

Page Customizations

Here is an illustration of the window presented:

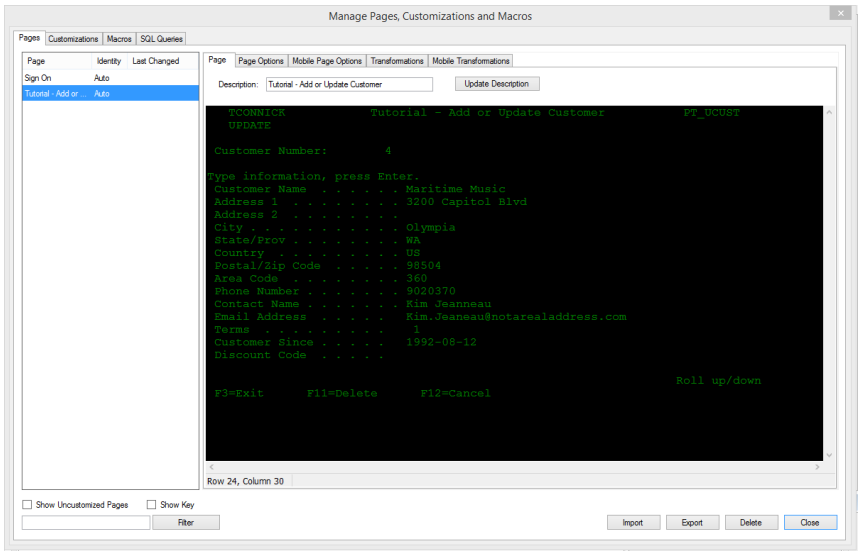


Figure 2-9. Presto's Manage Pages interface.

CHAPTER 19

Chapter 19 - Using the Responsive Visual Editor

About the Responsive Editor

Presto has historically used absolute positioning to retain the layout of the original green screen display. In this approach, Presto converts the green screen's row and column into HTML screen positions depending on the client's skin and font settings.

The disadvantage of this approach is that it doesn't always scale effectively to mobile devices. A mobile device might have a portrait-oriented screen that's too narrow to see all the elements if they are placed on one row. Several rows are necessary to fit the same number of elements that can fit on the same row on a desktop or laptop monitor.

In Presto, we solve this issue for mobile devices with a Responsive user interface that scales well for any screen resolution and orientation by using special HTML structure and CSS styles.

This Responsive user interface uses Bootstrap, a popular HTML5-based user interface framework. Bootstrap provides a flexible design to create responsive applications that automatically resize to fit the various screen sizes when you access them from different devices: http://getbootstrap.com/

Understanding the Responsive User Interface

The responsive user interface adapts to the viewing device, such as smartphone, tablet or desktop, based on the width of the screen. There are two layout formats, each utilizing the grid structure in a different way. In effect, a single set of customizations will adapt automatically to the device, without the need for separate layouts (such as different layouts for desktop and mobile).

The interface is responsive not only to different screen sizes in phones and tablets, but also to changes in aspect ratio when the users rotate their devices to work with the content more comfortably. When they do this, the UI can automatically resize to compensate.

Use the responsive mode to generate a responsive grid which changes its layout based on a threshold that is set to 768px. This means that when the browser's window size is smaller than 768px the stacked layout is applied:

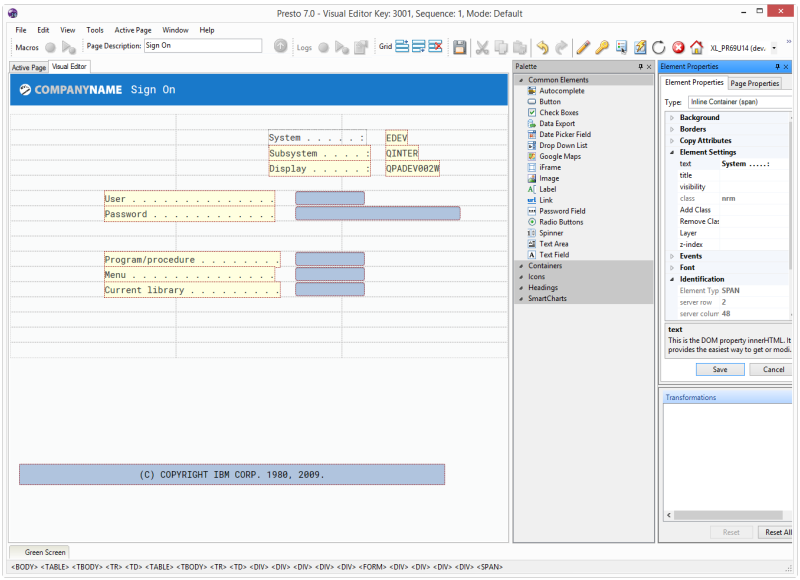


Figure 2-1. Working with the Responsive Editor.

The visual editor shows the grid by adding a dotted border around cells to give a better idea of the underlying HTML structure. In the Page View, these grid cells won't have any border.

The first time you open the page in the Visual Editor, the absolutely positioned elements are highlighted. The absolutely positioned elements are highlighted in yellow and blue (for input fields) for easier identification until you place them in the responsive grid.

You can drag and drop elements into individual cells, move elements between cells, and transform elements inside of the cell:

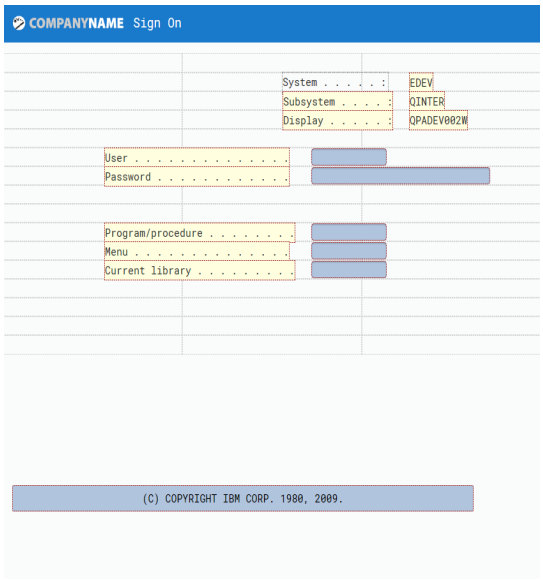


Figure 2-2. The grid of the Responsive user interface.

After elements are positioned and saved, the page will look exactly the same on a desktop as it does with absolute positioning:

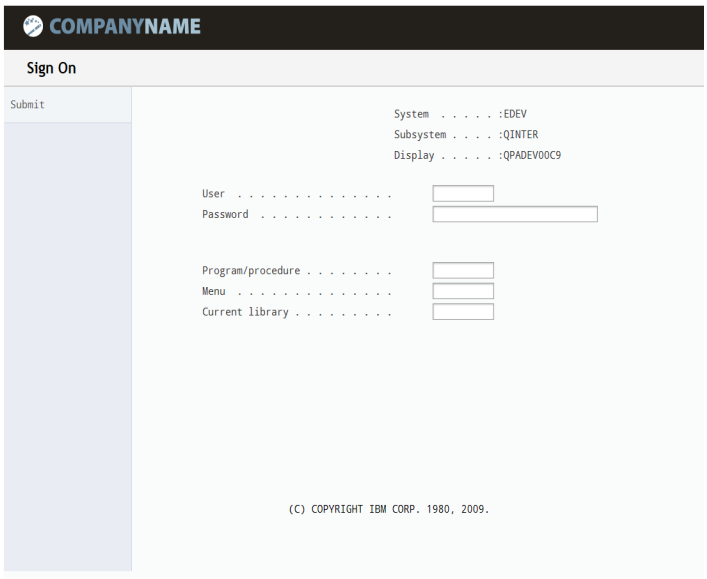


Figure 2-3. Viewing the Sign On Page on a desktop.

Presto defines a screen breakpoint of 768px at which the layout will change.

For phones and devices with a horizontal screen width less than 768px, columns switch to being stacked on top of each other. This avoids horizontal scroll bars for small devices that are easier to scroll vertically. As the screen width increases to greater than 768px (desktop devices), the columns change to a horizontal side-by-side layout.

When you run the program on a device that is narrow than 768px, the elements are stacked:



Figure 2-4. Viewing the Sign On Page on a mobile device.

In this illustration, we are running in a mobile-device emulator and reduced the screen width below 768 pixels. At this screen width, the elements are stacked on top of each other rather than staying vertical with a scrollbar.

Activating Responsive Features

To enable the Responsive UI and activate the Visual Editor for it, follow these instructions:

1. Open your skin's layout.html file (Tools>Modify Skin>layout.html).
2. Just above the `</script>` tag, uncomment this line:

```
<script>
        Presto.SetDisplayMode('responsive');
</script>
```

3. Save and restart the IDE session.

This option will tell Presto to run in Responsive mode.



The Responsive Visual Editor is disabled on screens with custom HTML attached. Detach the custom HTML from the screen to use the Responsive Visual Editor.

Change the Width of the Grid

You can set the grid width to design for the screen size that best fits your needs.

To change the width, follow these instructions:

1. Open your skin's layout.html file (Tools>Modify Skin>layout.html).

Directly above the `</script>` tag, uncomment this line:

```
<script>
        Presto.SetEnvironmentSetting('ContainerWidth', '1260');
</script>
```

In this illustration, the container width is set to 1260.

Placing Elements in the Grid

There are a few ways to place elements you have selected in the responsive grid:

- **Drag & Drop.** The quickest & easiest way to place elements in the grid is by dragging & dropping them with your mouse. To do this, left-click anywhere within the area contained by the dotted outline, then drag and drop the element to the target grid cell. When you release the mouse button, you'll notice the element snap into the grid cell.
- **Drag & Drop Blocks of Elements.** You can use your cursor to draw a selection box around a group of elements on the screen, and place them into the grid cells.
- **Arrow Keys.** You can place one or more elements in the grid by clicking them and using the arrow keys to move them.

Note that you can use Ctrl+Click to select multiple items before placing them in the grid.

These are the things that happen when you place an element in the grid:

- The element is no longer highlighted in yellow as an absolutely positioned element.
- The position of the element in the grid is reflected in the transformations window.
- The element is assigned a grid class of col-sm-*.

Placing an Element: Drag & Drop Example

In this illustration, we have picked up an absolute-positioned element and are moving it over to the grid:

To position elements in the grid, you just need to drag & drop them. When you release the mouse button, the element snaps into its cell in the grid.

After you drop the element, you'll notice that Presto has made the following changes:

- **Properties.** The element properties are updated with the position of your element. In this case, Presto updates the grid's row and column position.
- **Transformation Record.** The bottom right of your screen displays the transformation.

Position Properties

The Position properties include the following attributes:

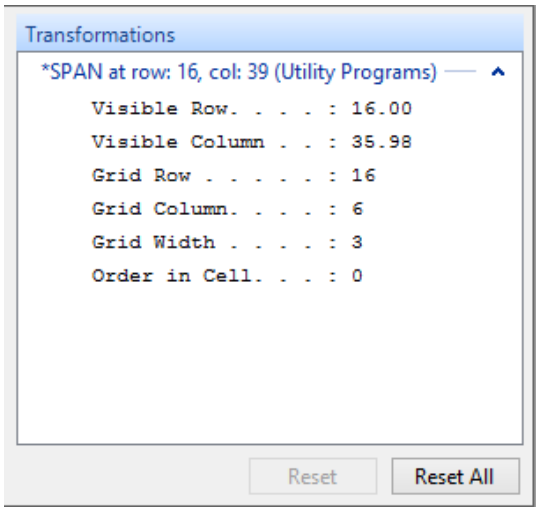


Figure 2-5. Grid Properties in Transformation Window.



An element's position in the Bootstrap grid is represented by the grid row and column values.

Resizing Elements

Presto assigns a default grid width to an element based on the amount of space it occupies on the original green screen display.

To resize an element start by selecting it. Then grab the tab in the lower right hand corner and adjust it to the height and width that you need.

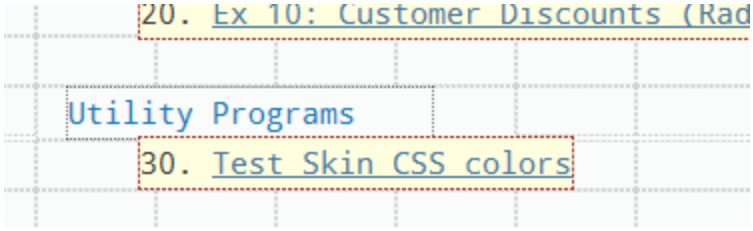


Figure 2-6. Using the element's handle to resize it.

When you do this, the grid-width attribute is updated to the transformation:

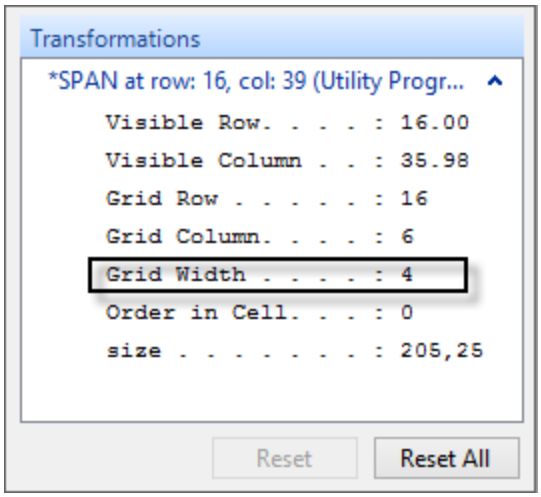


Figure 2-7. The Grid Width property is updated for the adjusted element.

Saving Visual Editor Changes

To help you avoid formatting issues, Presto checks that every element is placed in a grid cell. In other words, if your page still has some absolutely positioned elements, Presto will issue a warning message alerting you, and recommend you add the further elements to the grid:

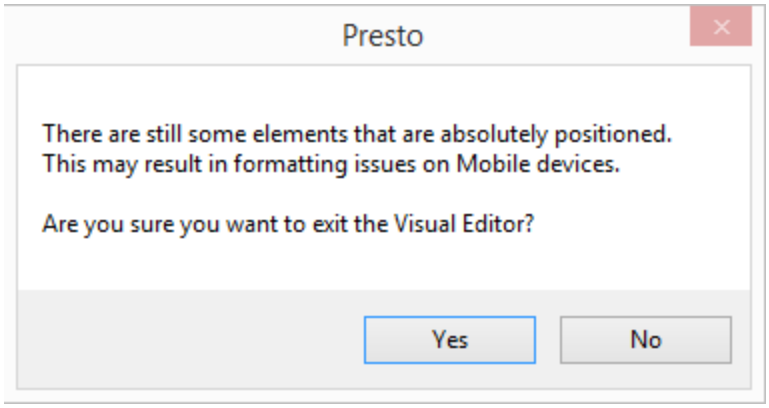


Figure 2-8. Saving your changes.

Adding Responsive Grid

Once you've enabled responsive display mode, the Presto Visual Editor will allow you to insert a Responsive Grid.

To add a Responsive Grid to the screen, right click anywhere on the page and this context menu will appear:

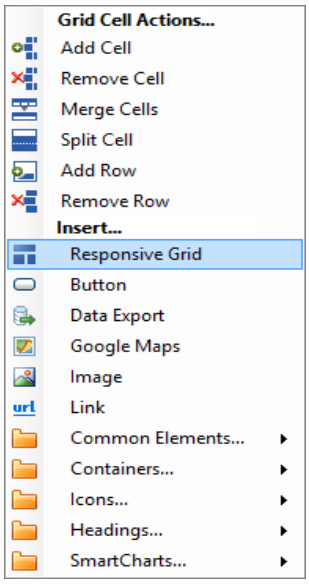


Figure 2-9. Inserting the responsive grid.

Click the Responsive Grid menu option, which displays a Create Grid dialog:

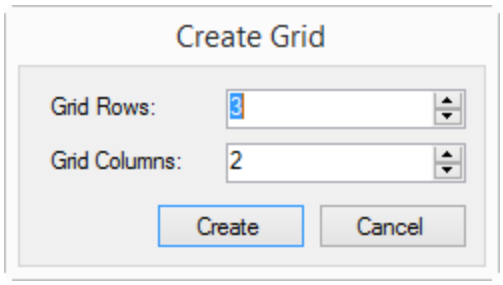


Figure 2-10. Selecting the number of rows and columns in the grid.

Enter the number of Rows and Columns for your grid. Then click the Create button, which will add the grid to the screen:

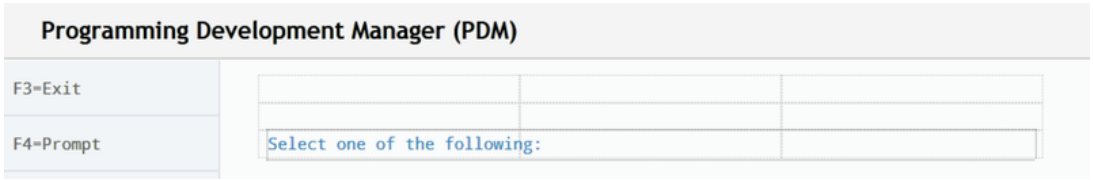


Figure 2-11. Inserting a grid with 3 rows and 3 columns.

Adding a Cell

While modifying the grid structure, you might need to add a new cell.

Select a cell or a row and right click to open context menu:

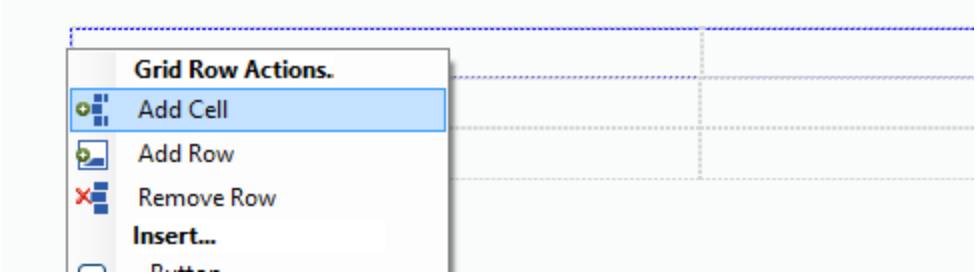


Figure 2-12. Adding a Cell.

Clicking on the Grid Cell/Row Actions>Add Cell which will display the Cell Settings dialog:

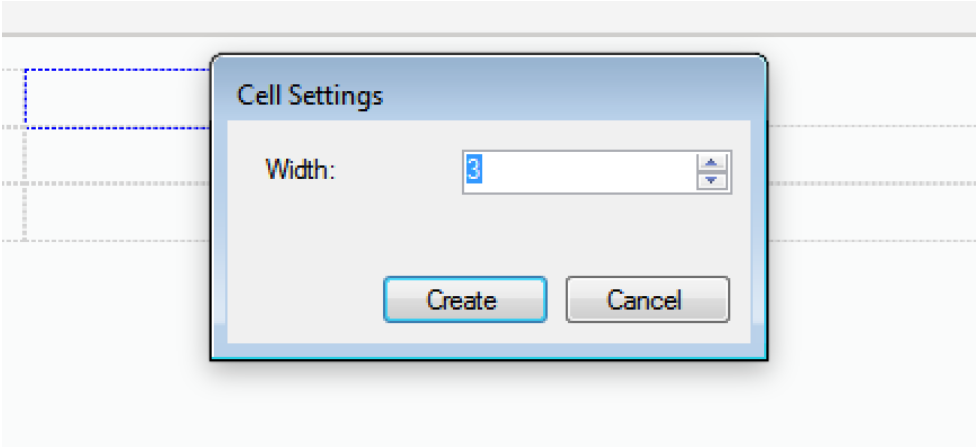


Figure 2-13. Setting the width of the cell.

In the above illustration, you can select the width of the new cell.

Click the Create button and the new cell will be added.

Adding a Row

You can extend the responsive grid by adding one or more new rows. To add a new row, click on a cell before or after where you want to add the new row:

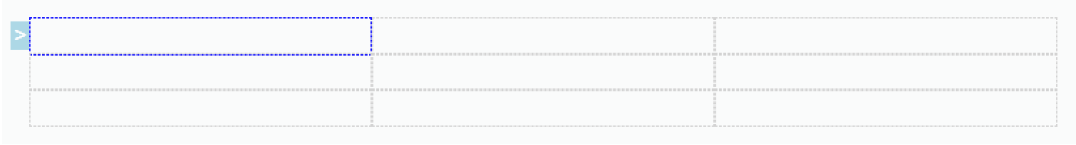


Figure 2-14. Selecting where to add a Row.

Right click the cell to open the context menu and click Grid Cell/Actions->Add Row:

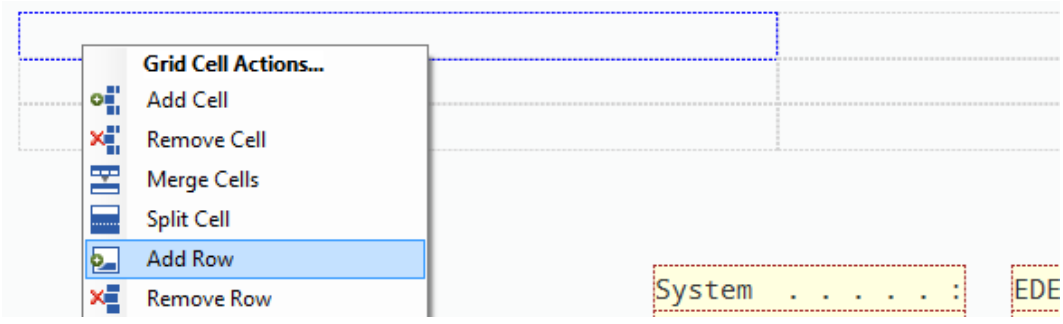


Figure 2-15. Adding a Row.

This displays the Row Settings dialog, allowing you to choose the number of rows and columns you want to add and where to add them:

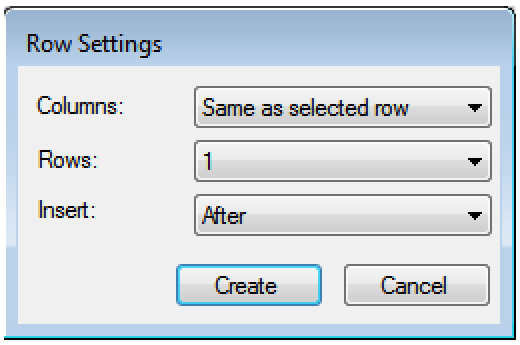


Figure 2-16. Row Settings Dialog.

You can choose to add the same number of columns as the selected row or from 1 to 12 columns:

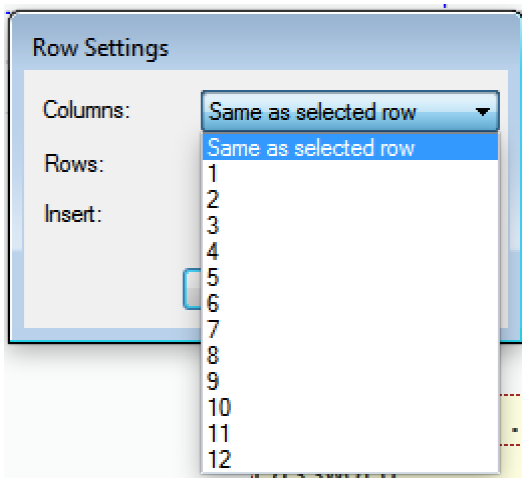


Figure 2-17. Selecting number of rows.

You can add one or up to 20 rows at a time:

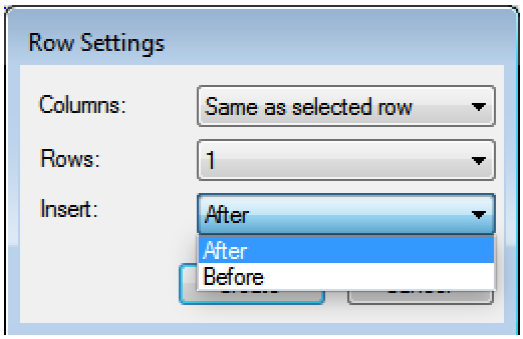


Figure 2-18. Inserting before or after.

The result will look like this:

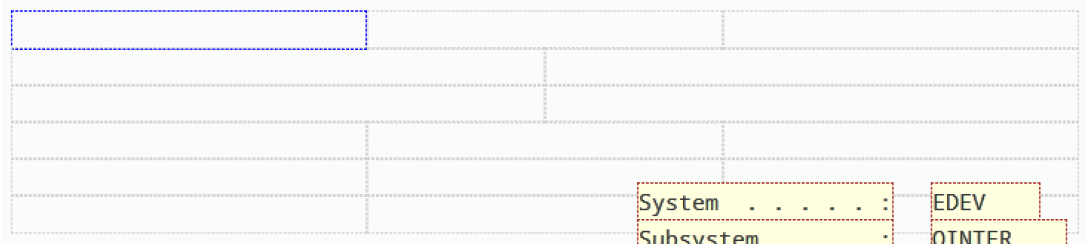


Figure 2-19. Result of inserting two rows.

Merging Cells

In Presto, you can merge two or more cells into one cell.

Select multiple cells by clicking on them while holding CTRL key.



Figure 2-20. Selecting a cell.

Right click to get the context menu:

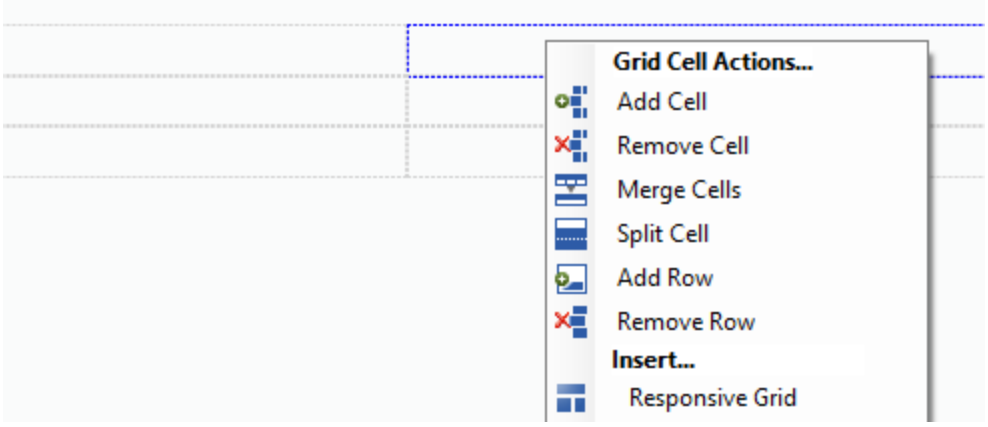


Figure 2-21. Context menu for Concatenating Cells.

Select the option Merge Cells:

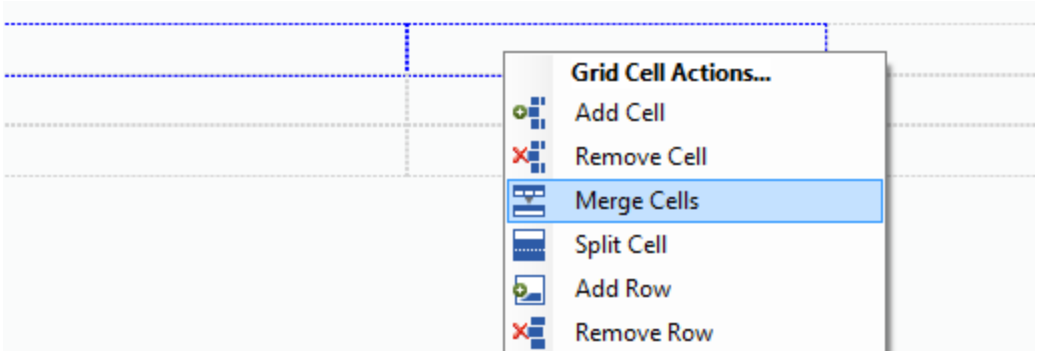


Figure 2-22. Option to Merge Cells.

The result is that the cells merge into a single cell:



Figure 2-23. Viewing the results of merging the cells.

How to Select a Row

To select an entire row, hover over a cell:

Figure 2-24. Preparing to select a Row.

When you hover over the cell, it displays an arrow on the left. Click this arrow to select the row:

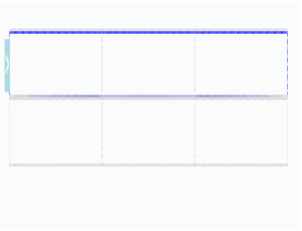


Figure 2-25. Selected Row.

Once you select the row, you can right click on the arrow on the left to display:

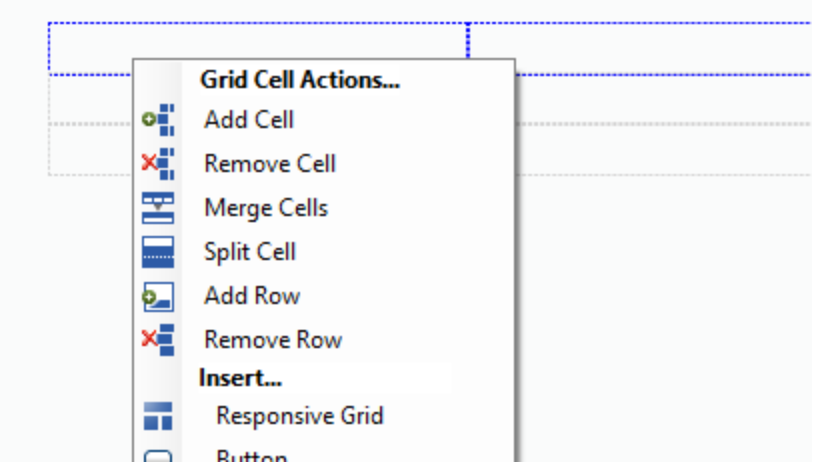


Figure 2-26. Grid Row Actions.

Removing a Cell

To delete a cell, click on the cell you want to delete, highlighting it with a blue border:



Figure 2-27. Selecting a Cell to Delete.

Right click to get context menu:

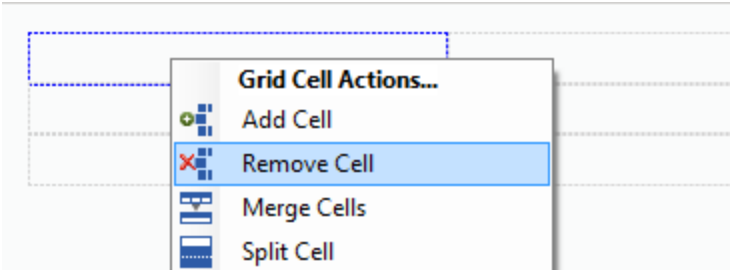


Figure 2-28. Remove Cell Option.

Click on remove cell and the selected cell will be removed.

Removing a Row

To remove a row, select a cell or row by clicking on it.

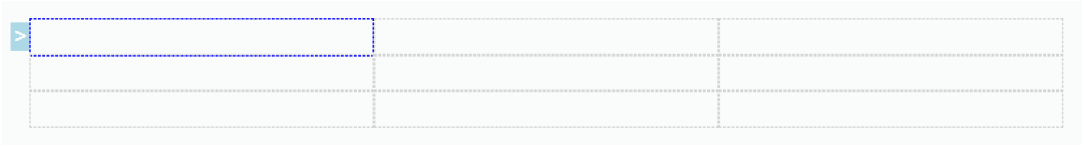


Figure 2-29. Selecting a Row to Delete.

Right click to open the context menu:

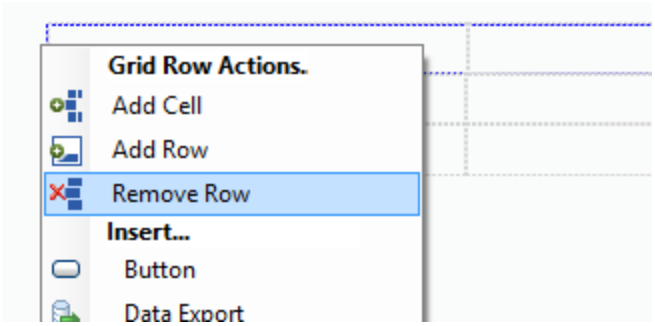


Figure 2-30. Option to Remove Row.

This option removes the selected row:

Figure 2-31. The result of removing a row.

Splitting a Cell

You can use the Split Cell feature to split one cell into two or more new cells. Select the cell or cells you want to split:

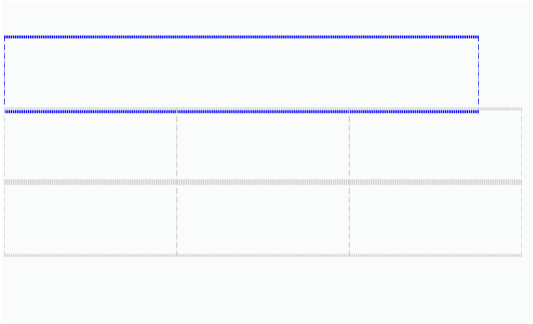


Figure 2-32. Selecting a cell.

Right click to open context menu and choose Split Cell:

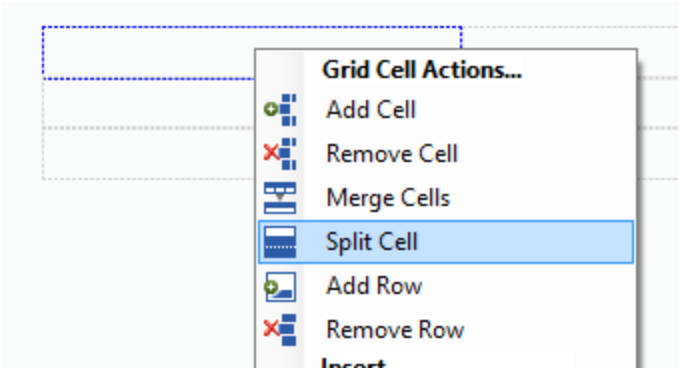


Figure 2-33. The Split Cell Menu option.

After splitting into multiple cells of one-column width, this is the result:



Figure 2-34. The results of splitting the cells.

Visible On Property

Presto gives you the option to only show cells or rows on selected device sizes.

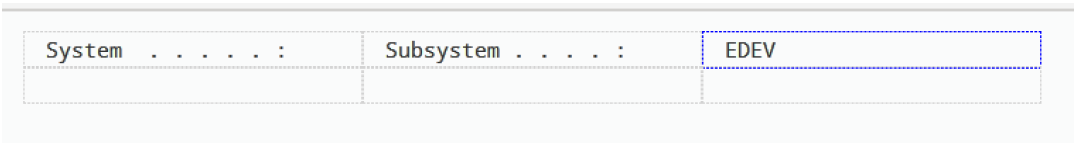


Figure 2-35. Selecting a Cell.

Select the cell and then in Element Properties choose Visible On and select one or more options:



Figure 2-36. Display the cell on devices with the selected screen size.

Then after saving the Visible On property, the selected cell will only be displayed on the chosen device size:



Figure 2-37. Viewing the results.

Hidden On Property

You can modify your grid to hide selected cells or rows on certain device sizes:

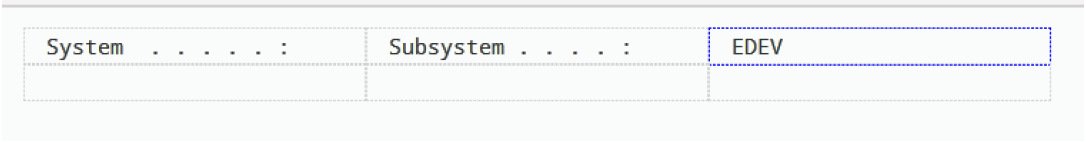


Figure 2-38. Selecting a cell.

Select the cell you want to modify:



Figure 2-39. Hide the cell on devices with the selected screen size.

Then, after saving, the selected cell will be hidden on those device sizes:



Figure 2-40. Viewing the results.

Placing an Element: Highlighting

When dragging an element into a grid cell, Presto highlights the elements before and after to show where the dragged element will be inserted:

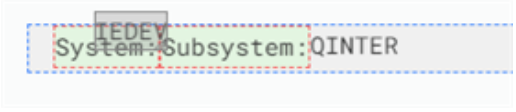


Figure 2-41. Highlighting the elements before and after the dragged element.

When you release the mouse button, the element is inserted between the highlighted elements in the grid cell:

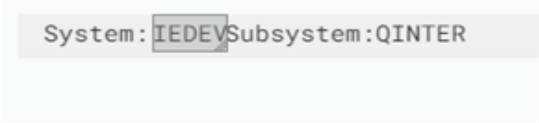


Figure 2-42. Element after insertion into grid cell.

About Creating Responsive Grids

When Presto builds the HTML for the individual grid cell, the code for your element looks something like this (using the user on the Sign on page as an example)

```
<div class="col-sm-5" grow="6" gcol="3" gwidth="5"><span id="C0617" style="position:
static; left: 0px; top: 0px;" class="nrm  transform-position-changed" row="6" col="17"
vrow="6.08" vcol="15.99" grow="6" gcol="3" gwidth="5" gorder="0">User  . . . . . .
. . . . .</span></div>
```

Notice the following item:

- The div tag is of class “col-sm-5”, which is tied to a CSS tag in the bootstrap.css controlling the appearance of the cell on the screen.
- The Bootstrap CSS class prefix “col-sm-” sets the grid cell to display horizontally when the screen width is greater than 768 pixels.
- Grid width is created by specifying the number of 12 available columns. For example, for a cell that spans 5 columns this class is “col-sm-5”.

Considerations

- Elements in the responsive grid can’t overlap.
- After you convert a page to the Responsive mode, you can’t open it in the original visual editor.

Additional Features

- Easy-to-use grid that will help you place the content of your custom pages.
- Backward compatible with absolute positioning.
- Fallback to absolute positioning when Responsive UI is inactive.
- Palette only contains elements supported by Responsive UI.
- Absolutely positioned elements are highlighted for easier identification.
- Responsive UI can be activated and deactivated at the skin level.
- Transformation window contains more properties to support the Bootstrap grid.

Responsive CSS Classes

Another useful feature, available through Bootstrap, is the responsive utility classes that can provide a simple way of showing or hiding elements at different screen sizes.

You can use the following responsive CSS classes to show an element on screens falling between the specified ranges:

- .visible-xs-*
- .visible-sm-*
- .visible-md-*
- .hidden-lg-*



You can add these classes together to show the elements on multiple screen sizes. For example, you can use both the classes .visible-md-*. and .visible-lg-* on the same element to show it on medium and large screens.

Example

For example, if you want to only show an element on a larger device or desktop, you can add these classes to your element:

- visible-lg and visible-md

The result is that this element is shown only on medium and large screens.

In addition, you can use the following responsive CSS classes to hide an element on certain screen sizes:

- .hidden-xs-*
- .hidden-sm-*
- .hidden-md-*
- .hidden-lg-*

Example

In this example, you can use a hidden responsive class to hide a table column on extra-small screens. You can add this class to your elements:

- hidden-xs

The result is that this column is only hidden on extra-small screens, and shown on all other size screens.



Please visit http://getbootstrap.com/css/#responsive-utilities for further information on responsive design using Bootstrap's responsive CSS classes.

Section 4 - Managing Applications

CHAPTER 20 - PRESTO CHANGE MANAGEMENT 379
Customization Categories 380
Overview of the Migration Process 380
Managing Environment Level Enhancements 381
Skins 381
JavaScript Enhancements in custom.js 382
SQL Queries 383
Recorded Macros 383
Site Configuration Settings 383
Managing Page Level Customizations 384
Environment Configuration Settings 384
Exporting Customized Pages 384
Importing Pages to Another Environment 386

CHAPTER 20

Chapter 20 - Presto Change Management

As we have said previously in this guide, Presto has two core components:

- A tool to automatically convert your screens to web pages.
- A tool for customizing and enhancing these web pages.

If you are only using Presto for its out-of-the-box web enablement features then your change management process will essentially consist of managing upgrades - creating and testing cloned environments, then toggling these live. If this is how you use Presto, you can follow the instructions in that describe how to clone and upgrade environments and switch from one active version to a different one.

This chapter is for everyone else - developers using the Presto design tools to further enhance and modernize applications. If you are using the visual editor or custom HTML pages, adding Smartcharts with associated SQL queries, customizing your environment's JavaScript, or in any other way enhancing your applications with additional features, read on!

Customization Categories

Presto allows you to enhance and extend your green screen applications in many ways. Broadly speaking, however, the changes you make fall into two categories:

- environment wide
- page specific

The following sections discuss how to manage each of these types of changes.

Overview of the Migration Process

When you plan to move changes from development to production or QA/UA environments, the number of components you need to migrate depends on the changes you have made to your site.

Here is a summary of the ways in which you can customize your green screen applications with Presto:

- **Skin changes**, such as company logos, color definitions, and other CSS changes.
- **JavaScript enhancements in your custom.js file**, which apply to many or all the pages in your application(s).
- **SQL queries** (potentially used in multiple charts, drop down lists, autocompletes, etc.).
- **Recorded macros** that you have attached to buttons or otherwise made accessible to your end users.
- **Advanced site configuration settings**, such as regular expressions to identify function key patterns and menu options and your site detection settings.

  These are normally encapsulated in the config.bws file you change through the Presto Designer's Tools>Advanced Settings menu option.
- **Page Properties** (function key settings, formatting options).
- **Custom HTML pages**.
- **Page-level JavaScript enhancements**, such as custom date pickers.
- **Visual Editor changes** (also called transformations).

The first five items in the list above are stored at the environment level, while the last four changes only affect individual pages.

The rest of this chapter describes the process of migrating these kinds of changes from one environment to another.



The suggested process below assumes that you are making changes in a development environment, then pushing those changes either to a QA/UA environment or a production environment (where your end users are logging in).

If you are making simultaneous changes in multiple environments, you will need to take that into consideration when reviewing the instructions below and implementing your own change management procedures.

Managing Environment Level Enhancements

Some of the modernizations strategies you will choose involve changes that affect your entire environment. These include:

- Skin changes.
- JavaScript enhancements in your custom.js file (and other resources you define, such as JS plugins, images, css files, etc.).
- SQL queries.
- Recorded macros.
- Site configuration settings.

This section describes how to move these components between environments.



For instructions on how to create new Presto environments please refer to the installation instructions in “Installation and Configuration” on page 7, and in particular Step Step 2, Install the Presto IBM i Components. describes how to create new or cloned environments,

Skins

Presto skins are sets of CSS and HTML files that reside in an IFS folder in the directory for a particular Presto environment (for example, `ESDI/XL_PRESTO/skins/CompanyX`, where `XL_PRESTO` is the name of your default development environment, and `CompanyX` is the name of your custom skin.

To move a skin from one environment to another you must:

1. Copy the IFS folder.
2. Set the new environment to use that skin (if it isn’t already).

You’ll normally only have to perform Step 2 the first time you change an environment to use a particular skin, or when you create a new environment.

When you make subsequent updates or changes to your skin in your development environment, you only have to copy the IFS folder from your working directory to your production directory.

As an example, your working IFS directory might be:

```
ESDI/XL_PR88DEV/skins/CompanyX
```

And your production directory might be:

```
ESDI/XL_PR88PRD/skins/CompanyX
```

To check which skin you are using in your current environment open your settings from the Tools menu, and select the Global Settings tab:

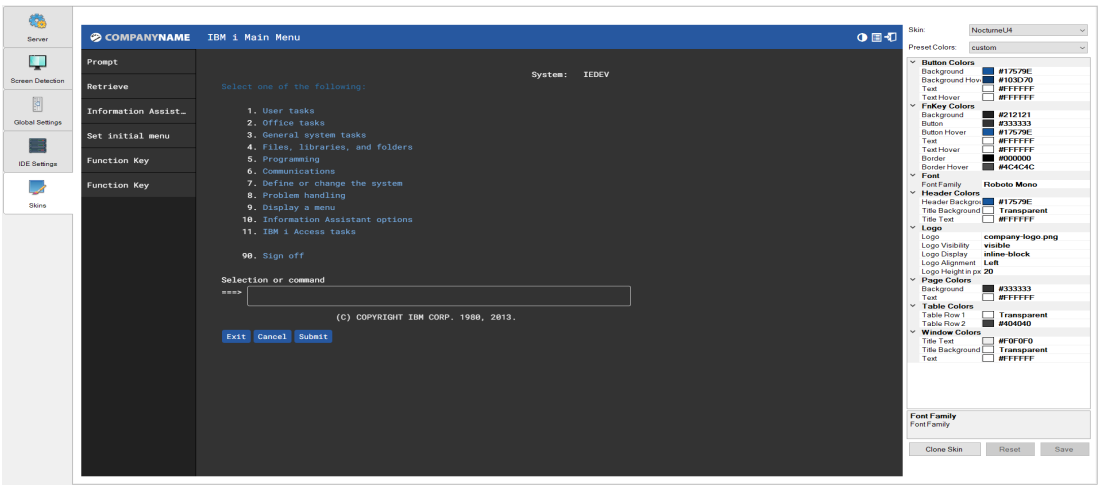


Figure 2-1. Confirming the currently active skin.

In this illustration you can see that we are using a skin named NocturneU40.

If we look in our IFS, this is what we find:

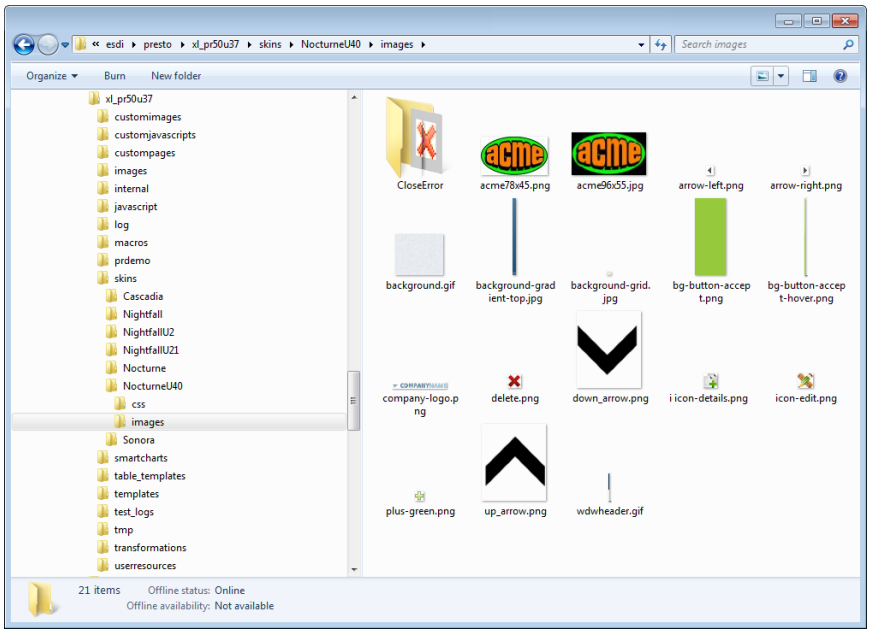


Figure 2-2. Skin IFS files.

To copy a skin from one environment to another for the first time, just copy the appropriate IFS directory from one Presto environment to the other.

On subsequent copies, you can rename the old directory in the target environment, or just replace all the files it contains with the updated ones.

JavaScript Enhancements in custom.js

Another type of change you may make at the environment level is to add JavaScript in custom.js.

The JavaScript file custom.js is in the folder `/ESDI/XL_PRESTO/userresources` (where `XL_PRESTO` would be the name of your development environment). You can see this directory from our example environment at the very bottom of Figure 2-2.

To migrate changes from one environment to another, simply copy custom.js from the /userresources/ directory in your development environment to your target environment.

If you have defined other custom user resources, you can copy this entire directory.

SQL Queries

You can use SQL queries in Smartcharts, dynamic drop down lists, autocompletes and data export/download features (among other possibilities).

Your SQL queries are stored as records in an IBM i file in your Presto environment. The simplest way to manage your SQL queries between environments is to (1) restrict your changes to your development environment, and (2) when you need to move your queries copy the entire file from one environment to the other.

Here is a sample command:

```
CPYF FROMFILE(XL_PR50U2/PR_QSTR) TOFILE(XL_PR50U21/PR_QSTR) MBROPT(*REPLACE).
```

The file that contains your SQL queries is named `PR_QSTR`, and in this example we are moving the SQL changes from an environment named `XL_PR50U2` to another named `XL_PR50U21`.



A Presto 'environment' is the same as a library. In other words, a discrete, self-contained instance of the Presto product.

Recorded Macros

Macros exist as named files in your environment's IFS directory (similar to skins and the environment-level changes in custom.js). If you have your environment's IFS directory open, you can see the macros folder a couple of lines above your skins folder (if you don't have your IFS open, you can see this in Figure 2-2).

To copy macros from one environment to another, copy all the files in that folder.

Site Configuration Settings

Your global site configuration settings are stored in a file named config.bws, in the root of your environment's IFS directory.

Moving page-level changes between environments requires that the global configuration settings be identical.

If you have made any recent changes to your configuration settings, copy config.bws from your development environment to the target location.

Managing Page Level Customizations

As you use Presto to modernize your application you may enhance individual pages by

- adjusting the page properties (e.g. function key settings, formatting options),
- making visual editor changes (also called transformations),
- adding page-level JavaScript enhancements, or
- creating custom HTML pages.

You can encapsulated and export these changes from Presto's Manage Pages window. In the new environment you can import them, overwriting the existing pages.

Environment Configuration Settings

Your site configuration settings should be identical between two related development and production environments. When you export a set of pages (as described in the sections below), Presto includes the environment configuration file config.bws in every export/import.



Note that some customers use separate environments for applications with significantly different purposes or design standards. Configuration files may vary between these types of environments.

Exporting Customized Pages

To export a page or set of pages start by selecting the Tools>Pages menu option in the Presto Designer. Here is an illustration from an environment with several custom pages:

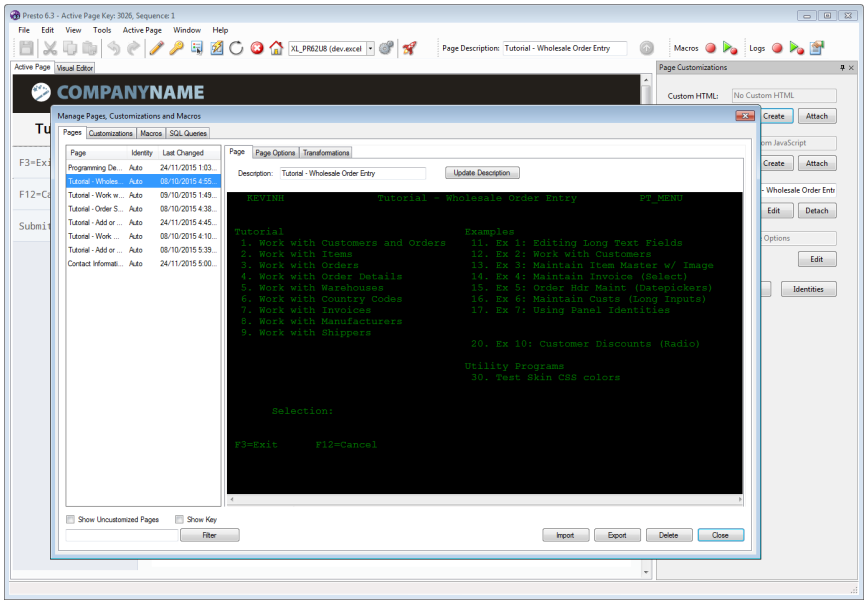


Figure 2-3. Preparing to export screens.

From the Manage Pages interface you can review the complete list of pages you have worked with in Presto, and their associated transformations. Click the Customizations tab to review a list you can sort by type, showing your HTML, JavaScript and Transformations.

In this illustration we have selected all the changed pages in this environment and are preparing to export:

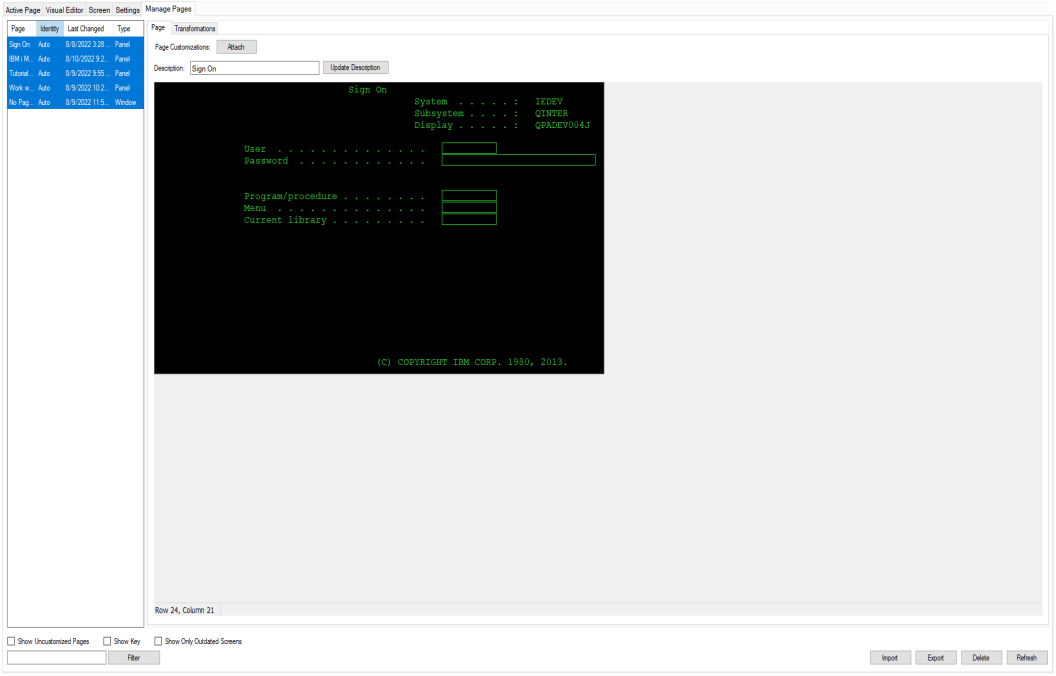


Figure 2-4. Selecting the pages to export.

You can hold down the Ctrl key while you click multiple items, or use Ctrl+A to select all your pages.

After you click the Export button, Presto presents a window confirming what you have selected:

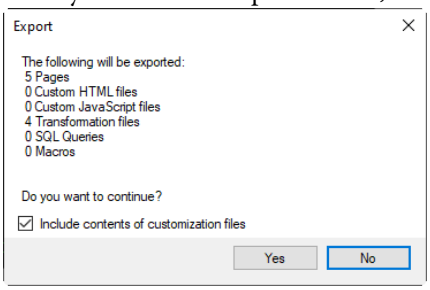


Figure 2-5. Export summary.

When you click Yes above, Presto prompts you to save the XML file containing all your pages and customizations:

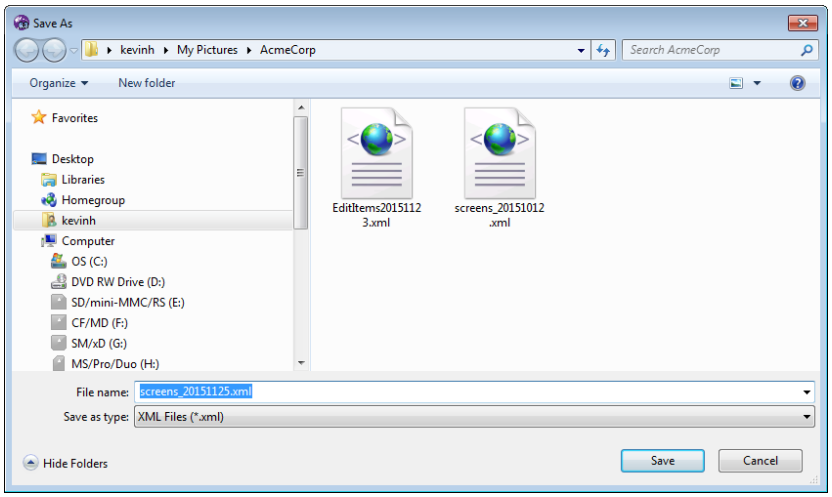


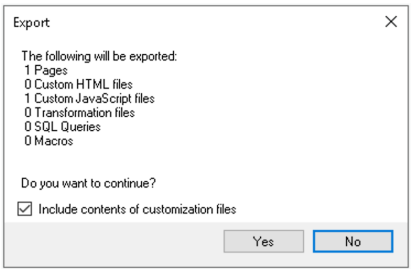
Figure 2-6. Saving the XML file with the custom pages.

The default file name is 'screens_YYYYMMDD.xml'. You can override the file name (for example, if you export multiple screens or sets of screens on the same day).

Save the file.

Exporting Customizations without the Contents

The option Include contents of customization files is available during the export process. See "Export Customizations without the Contents" on page 112 of Appendix 2



If this option box 'Include contents of customization files' is unchecked, the environment configuration settings will not be included in the .xml export file.

Importing Pages to Another Environment

To import your pages to a new environment, begin by connecting to that environment with the Presto Designer. Then go to Tools>Pages and click the Import button:

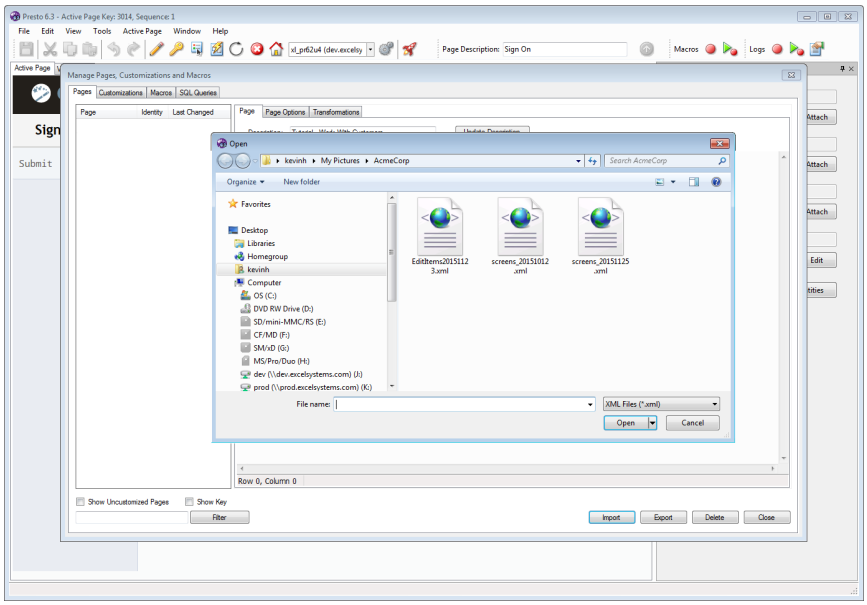


Figure 2-7. Preparing to import a set of custom pages.

In this illustration you can see that the environment we are connected to has no custom pages.

When you click the Import button Presto will prompt you to import an XML file.

When you select the file, Presto will spin for a second, and then display a confirmation message, then execute the import:

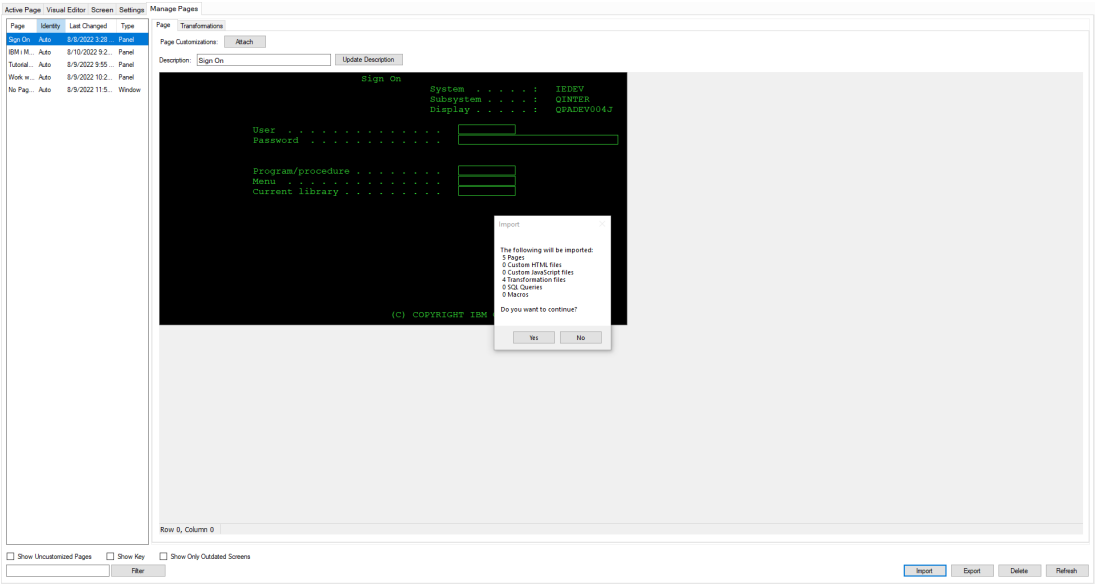


Figure 2-8. Environment after importing changes.

A 'Successfully imported' message confirms that all changes were copied into the new environment, and if you happen to be at one of the imported pages (as we are above), you should see those applied.



Remember that imported transformations may have other dependencies at the environment level (SQL queries, for example). Make sure you don't leave any components behind!

Failed Imports

If a customizations file cannot be imported for any reason, all changes are reverted and Presto creates a log file in the directory `/ESDI/XL_PRESTO/log` (replace `XL_PRESTO` with your own environment name).

You can attempt to interpret the log yourself (the critical error will likely be near the bottom), or contact Technical Support for help.

Section 5 - Appendices

APPENDIX A - ADDITIONAL PRESTO INSTALLATION OPTIONS 394
Configuring Multiple Installations 394
Installing a New Version 394
Creating a New HTTP Server Instance 395
Creating a New Server Connection 395
Renaming a Presto Library 396
Uninstalling a Presto Library 396
Using WebFacing with Presto 397
Virtual Terminals and CCSID Settings 398
Configuring Presto for Multi-Language Support 399
Working With the Presto Apache HTTP Server Instance 401
APPENDIX B - ADVANCED TECHNICAL TOPICS & TROUBLESHOOTING ... 403
Troubleshooting Presto Communications Issues 403
Rebuilding the PR_CONVQ Data Queue 404
How to Capture a Presto Debug Log 406
Designer Access to Server Logs 406
The Presto IBM i Jobs 407
Activating Polling Support 408
External Validation of Presto Session ID & User 409
Customizing Function Key Text Recognition 410
Subfile Option Flyout Customizations 411
Subfile Option Separator 411
Setting Detection Exceptions 411
Datepicker Auto-Detection 412
Presto Launch and URL Parameters 414
Custom Subsystem 416
Access to Page-Level JavaScript customizations in Firefox and Chrome Developer Tools 417

Multiple Data Queues 418
Bypassing Display Program Messages Screen 418
Load Testing with JMeter 419
How to create a Custom Error Handler 419

APPENDIX C - PRESTO.TRANSFORM FUNCTION REFERENCE 421

The Transform, Insert and Select Methods 421
Presto.Transform Support for Cascading Dropdown Lists 422
JavaScript Objects Used for the Transform and Insert Methods 423
Transformation Properties 425
to_object, object 425
prefix, suffix 429
source 430
value 431
Any Valid HTML Attribute 431
src 432
css 432
options 432
text_pos 433
Events Property 433
Presto Autocomplete Transformation Properties 435
Presto menulinks Transformation Properties 436
target 436
hide_value 436
hide_target 437
pattern_label, pattern_value, pattern_label_group and pattern_value_group 437
format 438
Customizing MenuLinks 438
Presto clickableoptions Transformation Properties 439
keep_target 439
Presto fileupload Transformation Properties 440
Presto Flyout Transformation Properties 441
JavaScript Functions to Retrieve and Set Presto Variables 442

APPENDIX D - PRESTO.EXECUTESQL 444

Technical Details 444
Parameters 444

Presto.OpenSession & Presto.CloseSession 446

APPENDIX E - PRESTO JAVASCRIPT API 448

Instance Methods 448

Constructor 448

AutoLogin 449

OpenSession 449

EndSession 450

SubmitRequest 450

PlayMacro 450

Window-Specific Methods 452

Setting and Getting the Window Title 452

Setting and Getting the Window Position 452

Setting and Getting the Window Size 452

Modifying and Working with Windows 453

OA-Specific Methods 454

SetFieldValue 454

SetSubfileFieldValue 454

Static Methods 454

GetConstant 454

GetField 455

APPENDIX F - INSIGHTS INTEGRATION 456

APPENDIX G - LICENSE KEY SERVER 464

License Server for Number of Users 464

License Server for Unlimited Seats 465

Presto Administrator Interface 465

Seat Count Utility 467

APPENDIX H - MDCMS CHANGE MANAGEMENT INTEGRATION 469

How to Configure MDCMS to work with Presto Change Management 469

How to use MDCMS in Presto 474

APPENDIX I - HEALTH CHECK TOOL 478

Object Status Reported 478

Non-Object Statuses 479

Running from the Command Line 479

Running the Tool via HTTP Request 480

APPENDIX J - FREQUENTLY ASKED QUESTIONS 482
Question and Answers 482
APPENDIX K - WEB PAGE DESIGN FOR GREEN SCREENERS 489
IBM i Screen Design vs. Web Page Design 489
HTML: The Language that Isn't 491
Writing Your First Web Page 492
Formatting Text - Simple Tags 495
Placing Images on Pages 498
Using Color in Your Pages 499
Navigating Using Links 500
Defining Tables 502
Interactive Pages - Using Forms and Forms Controls 507
Forms 507
Text Input Fields 508
Code that Affects the Entire Page 511
JavaScript 511
Style Sheets 512
Meta Tags 513
CSS (Cascading Style Sheets) 515
Why Use CSS? 515
Why You Should Use StyleSheets Now 515
Using Inline Style Attributes 516
Using Inline Stylesheets 516
Using Links to External Stylesheets 516
Stylesheet Media Types 517
Using External StyleSheets to Change the Entire Page's Appearance 517
Stylesheet Coding Rules 520
Using Comments 520
CSS Selectors 520
Element Selectors 520
Class Selectors 521
ID Selectors 522
Descendant Selectors 522
Examples of More Complex Selectors 523
Dynamic Pseudo-class Selectors 525
The CSS Box Model 526

'Block' versus 'Inline' Elements 526
The 'Cascade' in Cascading Style Sheets 526
Saving Style Sheet Specifications to External Files 528
CSS Best Practices 528
CSS Magic 530
How To Create Pseudo-Submit Buttons 531
Changing the Appearance of Forms Controls 534
Hyperlinks 535
The <DIV> and <SPAN> Tags 536

APPENDIX L - SOURCE CODE COPYRIGHTS AND ATTRIBUTIONS 537

Scintilla and SciTE 537
Firebug 537
Blowfish 538
Newtonsoft.Json 538
CRCpp 538
Apache License 539

Appendix A - Additional Presto Installation Options

This appendix describes some additional installation and configuration options available.

Configuring Multiple Installations

You can install multiple versions of Presto on your system, either to apply different environment defaults to particular applications, or to have Presto test and development environments where you can experiment with new skins and other features without affecting your current production applications. You can run the versions via separate HTTP server instances.

This section describes how to install and set up multiple Presto environments, and clone existing environments. The Presto Upgrader's Guide also provides additional information and guidance on creating cloned or test environments.

Installing a New Version

To install a new, unique version of Presto re-run your IBM i side install, following the instructions in the Presto Upgrader's Guide.

When prompted, choose a new, unique name for Presto environment library (different from the default value `XL_PRESTO`). Note that the does not have to exist.



A good choice for your library name is to include the intended port number of your new HTTP server instance in the library name (for example, `XL_PR8027`).

You can accept the rest of the default server side install options.

You won't need a separate version of the Designer; in the Designer you will just create a new server connection after creating the HTTP server instance.

Creating a New HTTP Server Instance

Every unique installation will also have a corresponding Apache HTTP server instance. By default, a second instance will use port number 8023. If you are creating more than 2 environments, you need to change the port number to a unique value. Follow these steps:

1. At a web browser, navigate to the Admin page for your HTTP servers. Find the server instance with the same name as the environment you specified.



For details on how to run the Admin interface for your HTTP configuration and edit your new instances, you can review the section "Working With the Presto Apache HTTP Server Instance" on page 401, below.

---

2. Edit the instance configuration to use a port other then 8022 (the default port for Presto). A good choice is 8023, and so on.

After saving your changes, restart the server instance.

Creating a New Server Connection

Once you have installed the new IBM i library and configured a corresponding HTTP server instance, you can create a new server connection in the Designer.

In the Designer, select the Tools>Connections option and click the Add button. Your new connection should look something like this:

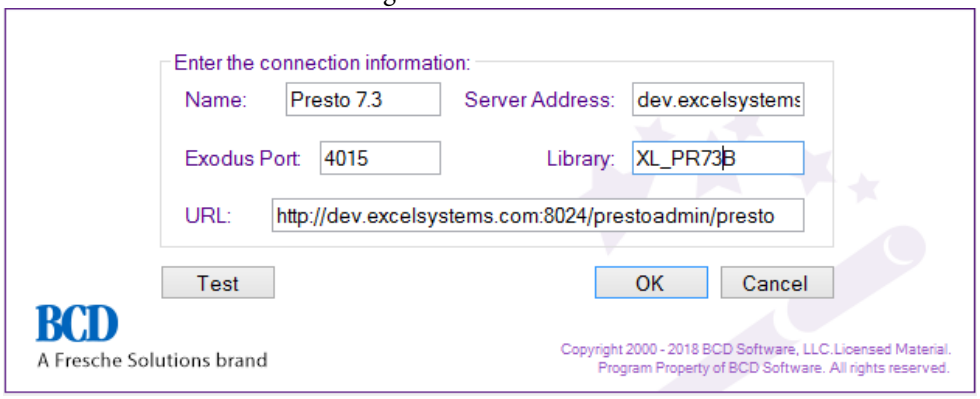


Figure 2-1. Adding a new connection on a different port.

This configuration includes two changes from your default:

1. The library is XL_PR73B, corresponding to the name we assigned to the second IBM i installation of the product.
2. The port number in the URL is 8024 (rather than the default of 8022), corresponding to the port designated in the new HTTP server config.

Click Test to verify your settings, then OK to create the new server connection.

Renaming a Presto Library

Presto ships with a command to rename a Presto library, `RNMPRLIB`, which you can use if you need your production library to have a particular name after an upgrade:

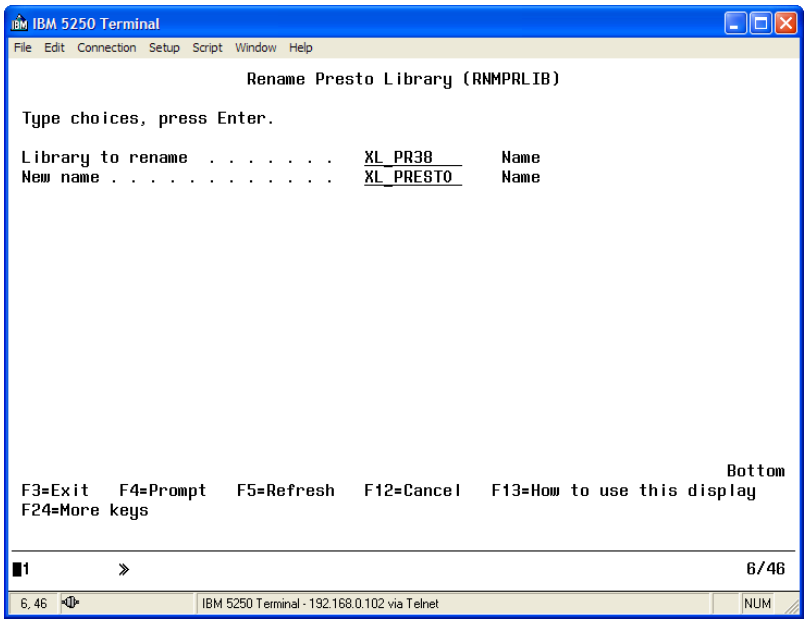


Figure 2-2. Running the Rename Presto Library command.

This command is described in our online Knowledge Base article 9436:

https://www.myfreschesolutions.com/kb?kbnum=9436

Please contact Technical Support if you have any questions. Your HTTP server instances are not touched by this process.

Uninstalling a Presto Library

Presto ships with an uninstall program to remove a Presto Library and all of its components, which you can use to remove old Presto environments after an upgrade. This command will remove the Presto Library, IFS folder, and HTTP Server instance.

You can execute the command in one of two ways:

1. `CALL XL_PRESTOI/UNINSTALL`
2. If you no longer have XL_PRESTOI use `CALL XL_PR98/UNINSTALL`, replacing XL_PR98 with the name of your Presto Library. Please note that this method is only applicable for deleting other Presto environments; it cannot delete the library you are currently using to run this program.

In the next screen, you will be prompted to specify the library name for the Presto environment you wish to uninstall.

Using WebFacing with Presto

If you have the WebSphere V6R0 Application Server installed you can enable Presto's option to use WebFacing. By using WebFacing you can avoid using interactive resources for your Presto sessions.



The only WebSphere component that Presto needs to have running is the WebFacing server. The WebSphere application server is normally installed on most IBM i systems - just make sure you have all the latest PTFs.

You can find additional information in the System Requirements listed inside the front cover, and at the IBM website, at http://www-03.ibm.com/systems/i/software/websphere/services/servicews60_V5R4.html (in the PDF version of this guide you can click this link). You can determine if you have the correct version of the WAS by running the `GO LICPGM` command and taking an option 10. You should see an entry like this:

```
733W60 *INSTALLED WebSphere Application Server V6 Express
```

To use WebFacing you need to check that option in the server settings (see Step Step 9 of) and start the WebFacing server instance.

Starting the WEBFACING Server Instance

Run this command to start the `WEBFACING` server instance:

```
STRTCPSVR SERVER(*WEBFACING)
```

To verify that the `WEBFACING` (Presto) server has been started correctly run the `WRKACTJOB` command and look in the `QSYSWRK` subsystem for the following jobs:

- `QQFVTSVR`. The WebFacing virtual terminal server.
- `QQFWFSVR`. The WebFacing run-time server.

If you do not see these jobs, the WebFacing server has not been started. You can run the command `WRKACTJOB JOB (QQF*)` to confirm this.



If you are having trouble starting the `WEBFACING` server instance check the system value `QAUTOVRT`. The value for `QAUTOVRT` determines the number of virtual terminal jobs that can be auto started. If this value is 0, then you cannot launch any browser sessions in place of VT jobs.

Set `QAUTOVRT` to a value that roughly corresponds to 3-5 times the largest number of simultaneous sessions you anticipate having connected to your IBM i). To change the value for `QAUTOVRT`, use the Work with System Values (`WRKSYSVAL`) command.

Virtual Terminals and CCSID Settings

When using certain languages (for instance DBCS languages) it is necessary that the virtual terminal session created have specific device description characteristics. In order to set the characteristics for the language you are choosing you need to add a section like this one to your internal Presto configuration file, config.bws.

```
<languages>
  <language type="default" charset="1173" codepage="833" workstationtype="10" />
</languages>
```

You can find config.bws in the root Presto directory in the IFS (esdi/presto).

You need to set the charset, codepage and workstationtype values. In this example the default language has been set to Korean.

How to Determine the Correct charset, codepage and workstationtype Values

To determine the correct values are for any given language follow these steps:

1. Start a properly configured Client Access session.
2. At the Sign On screen make a note of the device description (this is shown as the Display value):

```
                Sign On
                           System  . . . . . :   S1038EF2
                           Subsystem . . . . :   QINTER
                           Display . . . . . :   QPADEV0005

User  . . . . . . . . . . . . . .   _________
Password  . . . . . . . . . . . .
```

3. Log on to your session and run the command DSPDEVD for the device description you noted in Step 2. The displayed information will include a section similar to this:

```
Device class . . . . . . . . . . . :   *VRT
Device type  . . . . . . . . . . . :   3487
Device model . . . . . . . . . . . :   HA
Internet address . . . . . . . . . :   192.168.0.236

Online at IPL  . . . . . . . . . . :   *NO
Attached controller  . . . . . . . :   QPACTL01
Keyboard language type . . . . . . :   USB
Character identifier . . . . . . . :   697    37
```

Figure 2-3. Displaying the device description.

   On the last line in this block, the first part of the Character Identifier is the 'charset' and the second value is the 'codepage.'

4. In order to determine the workstation type you must cross reference the Device Type and Model shown above with the “Supported Work Station Types and Models” table near the bottom of this web page:
   https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/apis/QTVOPNVT.htm
   As an example, the device description in the illustration above (3487 HA) corresponds to a work station type value of 14.

Configuring Presto for Multi-Language Support

You can use the same Presto library and Apache instance to support multiple languages.

Apache HTTP Server Configuration Requirements

To do this, you need to add a section along these lines to your Apache config:

```
# Default CCSIDs
DefaultFsCCSID  00037
DefaultNetCCSID 01208

# Russian
<LocationMatch  /presto(admin)?/ru/>
   CGIJobCCSID  1025
</LocationMatch>

# Korean
<LocationMatch  /presto(admin)?/kr/>
   CGIJobCCSID  933
</LocationMatch>

ScriptAliasMatch ^/presto(admin)?/([a-z]{2}/)?presto$ /qsys.lib/xl_pr32.lib/presto.pgm
AliasMatch ^/presto(admin)?/([a-z]{2}/)?(.*)$ /esdi/xl_pr32/$3
```

In this example, XL_PR32 is the name of your Presto library.



Note that the ScriptAliasMatch line illustrated above needs to be a single line in your configuration; it only wraps here because of the size of this page.

Runtime Requirements

To use a particular character set at runtime you need to invoke Presto with a URL similar to these:

Language	Path
English	/presto/presto
Korean	/presto/kr/presto?lang=kr
Russian	/presto/ru/presto?lang=ru

To explain, in the Korean example, the first 'kr' (in the path) is used by the Apache server to set the right CCSID at that stage. The second 'kr' (the language code parameter) maps to a type that is used internally by Presto to set the correct character set and codepage, based on the values in your config.bws (the two letters map to the 'type' value in the XML shown below):

```
<languages>
  <language type="kr" charset="1173" codepage="833" workstationtype="10" />
</languages>
```

We chose not to use the first 'kr' for both purposes because some clients want complete control over the path shown in the browser.

You can find config.bws in the root Presto directory in the IFS (esdi/presto).



Note that the second line illustrated above needs to be a single line in your configuration; it only wraps here because of the size of this page.

You need to set the charset, codepage and workstationtype values. In this example the language has been set to Korean.

To determine the correct values for a given language, refer to the instructions in the section "How to Determine the Correct charset, codepage and workstationtype Values" on page 398, above.

You need to add a line to config.bws for each language you intend to support which also requires a unique device type. This is how you would add multiple languages:

```
<languages>
  <language type="kr" charset="1173" codepage="833" workstationtype="10" />
  <language type="ru" charset="xxxx" codepage="xxx" workstationtype="xx" />
  <language type="fr" charset="xxxx" codepage="xxx" workstationtype="xx" />
        etc....
</languages>
```



Note that the <language> lines illustrated above need to be a single line in your configuration; they only wrap to a second line here because of the size of this page.

If you need any assistance with this feature please contact Technical Support.

Working With the Presto Apache HTTP Server Instance

The installation process is designed to automatically create an Apache web server instance called XL_PRESTO (though this will match the environment name you selected in Step Step 2 of the installation instructions). You can confirm that the HTTP server was started by running this command:

```
WRKSBSJOB SBS(QHTTPSVR)
```

You should see several jobs named XL_PRESTO (or your selected environment name) in the QHTTPSVR subsystem.



If this server instance is not visible you can start it using the command STRTCPSVR SERVER (*HTTP) HTTPSVR(XL_PRESTO). If you have any difficulty with this process, contact Technical Support.

The default configuration will normally get you up and running, though you may decide to make changes to it before promoting applications to production.

Working with the Presto HTTP Server Instance

The default configuration on install should work for you, but should you need to, you can make changes to the XL_PRESTO HTTP Server configuration via IBM's Web Administration for IBM i pages.

You can access this interface from your browser, at:

```
http://your_server_ip:2001/HTTPAdmin
```

Insert your IBM i IP address before the :2001). Log in using your IBM i user profile.



If when you attempt to access the admin interface you get a 'Page not Found' error, you may need to start the admin server from your IBM i command line: STRTCPSVR SERVER(*HTTP) HTTPSVR(*ADMIN).

To confirm the Admin server has started run the command WRKACTJOB and look for it under the QHTTPSVR subsystem.

The page presented includes a list of your HTTP servers in a drop down box. Find and select the Presto entry (the example below uses XL_PR28U, an internal environment for testing):

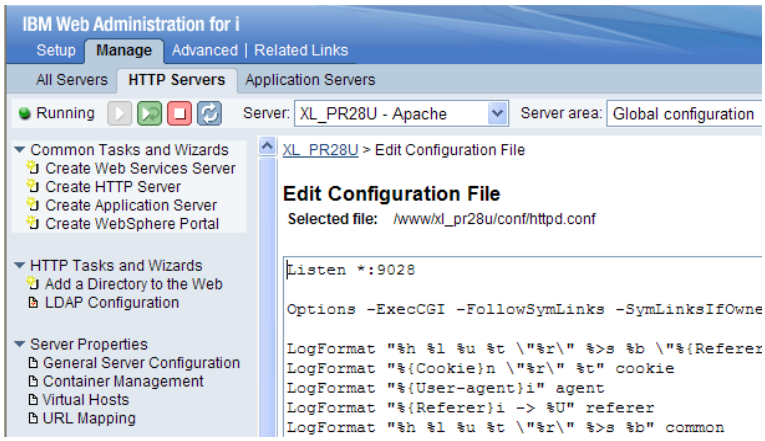


Figure 2-4. Working with your IBM i HTTP servers via the online interface.

If the server isn't running (i.e. the status is 'Stopped'), you can click the green button to start the server instance, changing the status to 'Running'.

Changing the Presto Server Instance Default Port Number

To avoid the possibility of conflicting with any existing HTTP server instances you may have running on your system, we ship the XL_PRESTO server instance configured to use port 8022. If you already have something running on that port, you can change this value in the configuration file.



If you aren't running any web applications on your IBM i, you can also run your Presto applications on port 80, which removes the need to include a port number in the URL.

Saving Server Configuration Changes

If you make any changes to your server configuration, you need to save those, then restart the instance.

If you have any questions about how to change your HTTP server configuration, or whether you need to use a different port number, please contact Fresche Technical Support.

Appendix B - Advanced Technical Topics & Troubleshooting

This appendix includes information on a variety of technical topics.

Troubleshooting Presto Communications Issues

If you are navigating through your application with Presto, and have a communication problem where your session dies, you can follow these steps to try to identify the problem:

Check Authorities on QTMHHTP1.

User profile QTMHHTP1 needs *ALL authority to QTMHHTTP's MSGQ. To determine if this is set, run this command:

```
WRKOBJ OBJ(QTMHHTTP) OBJTYPE(*MSGQ)
```

Use option 5 to display authorities.

Check that your system allows virtual terminal jobs.

QAUTOVRT is a system value that needs to be large enough that Presto can create additional virtual terminal jobs. Run this command to set this value:

```
CHGSYSVAL SYSVAL(QAUTOVRT) VALUE(*NOMAX)
```

While you don't necessarily need to set this value to *NOMAX, it should be at least 3-5 times as high as the largest number of simultaneous Presto sessions you anticipate having.

Is the QUSRWRK subsystem running?

Check to see if the QUSRWRK subsystem is active. You can run the command `WRKSBS` to see all active subsystems.

If it is not active you can start it with the following command:

```
STRSBS QSYS/QUSRWRK
```

The FTP TCP/IP Service Must be Started to Install the Software

In order to transfer the IBM i install components from the PC to the IBM i, you must have the FTP server started. Run this command if it is not:

```
STRTCPSVR SERVER(*FTP)
```

If for some reason you are not able to do this, check our online knowledge base for instructions on how to transfer the install files to your IBM i manually.

Is the Presto APACHE HTTP Server Instance Running?

In order to run Presto, you must have the XL_PRESTO HTTP server instance running (or another production server instance; if you have multiple environments, or have upgraded from an earlier release, your server instance may have a different name, such as XL_PR24.

Is the Exodus Subsystem Running?

In order to use the Presto 10.0 Designer, the Exodus55 subsystem must be running. Use this command to start the subsystem: STRSBS XL_EX55/EXODUS55.

Is the WebFacing Server Running?

If you are using the WebFacing server, it must be running. Run this command to verify:

```
WRKACTJOB JOB(QQF*)
```

You should see two jobs, QQFVTSVR and QQFWFSVR in the QSYSWRK subsystem. You can also use this command:

```
VRYCFG CFGOBJ(QQF*) CFGTYPE(*DEV) STATUS(*ON)
```

If they are not running, run this command to start them:

```
STRTCPSVR SERVER(*WEBFACING)
```

If when you reconnect after ending those jobs, re-starting a Presto session immediately sends the jobs back into TIMW status, there is a problem with the WebFacing server and it should be restarted. Run this command:

```
ENDTCPSVR SERVER(*WEBFACING)
```

Wait a minute. Then run this command:

```
STRTCPSVR SERVER(*WEBFACING)
```

Check the status of the Presto jobs.

Run the WRKACTJOB command on your IBM i and look for XL_PRESTO jobs in QUSRWRK. These jobs should be in a DEQW status. If they are in a TIMW status, they aren't working and should be ended. Then you can attempt to reconnect.

WebFacing Fatal Error: WebFacing User Job Not Started

If you get an error that the WebFacing user job was not started, the possible causes are:

- the maximum number of virtual devices has been reached, or
- the maximum number of jobs for the interactive subsystem has been reached.

To address this you can end and restart the WebFacing server.

You can further information on this topic in the section "Using WebFacing with Presto" on page 397.

Rebuilding the PR_CONVQ Data Queue

There are circumstances under which the PR_CONVQ data queue in your working Presto environment can become either full or corrupted. If this happens, you can run the command CLRDTAQ, in your current Presto environment, to recreate this data queue.

The error we most frequently see in this context is the following:

```
Storage limit exceeded for data queue PR_CONVQ in XL_PRESTO.
```

The library name may vary.

Some systems seem particularly prone to issues with this data queue. If you find that the `PR_CONVQ` data queue is filling up or becoming corrupted on your system, please contact Technical Support to discuss this, and so we can get a sample of your data queue from you.

How to Capture a Presto Debug Log

If while running your programs in Presto there is a consistent spot at which you get an error (i.e. for programmers, a reproducible error), you can turn on the debug flag in the Presto Designer, reproduce the problem, and then send the spool file that is generated to our development team for diagnosis.

You can find instructions for creating a Presto debug log in our online Knowledge Base article 7558:

https://www.myfreschesolutions.com/kb?kbnum=7558

Email this log file as an attachment to BCD Technical Support. You can also call us and we can walk you through this process.

Designer Access to Server Logs

If you encounter an error in Presto and the support team requests your server logs, you can now access these easily via the Help menu (Help>Server Logs):

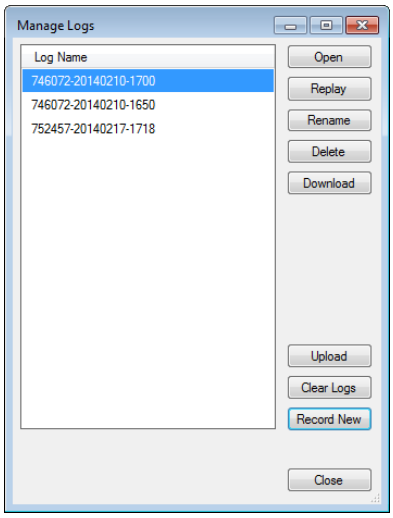


Figure 2-1. Reviewing server log files.

Double-click a file to open it in the Presto Designer. Use the Clear Logs button to remove the files from your system.

The Presto IBM i Jobs

There are 4 jobs that may be running when a Presto job is active:

1. If you have an Designer session up, there should be an EXWORK job in the EXODUS55 subsystem.
2. In QUSRWRK there will be a job that looks like the following:

```
XL_PRESTO QTMHHTP1 BCH .0 PGM-PR_HUB DEQW
```

   This is the job that communicates with the WebFacing server and handles the interpretation of the data passed back from the WebFacing server. We refer to this as 'The Hub'.

3. The browser will communicate with the Apache HTTP server instance server (by default named XL_PRESTO), but there will not be one entry for each session.

WebFacing

If you are using WebFacing, there will also be a WebFacing interactive job for every Presto session you have active. The best way to find this is by running the command: WRKACTJOB JOB (QQF0*). This job will look something like this:

```
QQF0426A5G LARRYF INT .0 CMD-ZWRKPDF DSPW
```

These WebFacing jobs provide the data to the hub. If you want to find the following jobs:

- QQFWFSVR (the WebFacing run-time server), or
- QQFVTSVR (the WebFacing virtual terminal server)

run the command WRKACTJOB JOB(QQF*) or look in QSYSWRK.

Activating Polling Support

If you have IBM i programs that normally 'push' messages and information to your screens, Presto is not by default configured to monitor for these to display them. However, there are settings you can access to enable this functionality.

This can be useful for cases where the IBM i screen is refreshed automatically (e.g. without sending anything to the server). Example include clocks, auto-run demo sequences, and `SNDBRKMSG` command output. Without this feature enabled, a break message would only be seen the next time the user initiated a server request. When the polling server is on, a JS function triggers an HTTP request every X seconds (as specified in the pollinterval value).



We don't ship the software with this option turned on because its use represents a substantial cost in network traffic, as well as processor on the IBM i.

If you need assistance with this feature please contact Technical Support.

About the Polling Server

You can enable and configure the polling interval from the Server Settings dialog (Tools>Settings).

Check the 'Poll Server' option to enable this feature, and then set your polling interval (the number of seconds between polls).

Turning On Polling for Individual Screens

You can also just turn this feature on just for a specific panel, from the Page Properties.

External Validation of Presto Session ID & User

In your Presto application you may choose to add code to reach out to other applications or databases. For example, one customer was using some custom AJAX to call various PHP programs.

However, these external calls are not protected by the Presto Session ID.

As a solution to this need, Presto includes a function that when given a Session ID, returns a Y/N response saying whether the session is valid, and what the current user of that session is. You can use this in your PHP programs to secure sensitive programs that need to know if a Presto session is legitimate. The current user is needed to make sure if Presto user A logs in, they cannot call the PHP programs pretending to be user B and so get access to information that they shouldn't be able to.

The task name is `validateusersession`.

Return Value Description

JSON Object	Possible results: • {result: 1} - Session is valid • {result: 0} - Session is invalid • {error: "", code: "", critical: ""} - If error occurred.

Parameter Descriptions

SessionId	Presto Session ID.
user	User Name (who is logged into the given session)

Example:

Here is an example URL:

```
PRESTO_URL?task=validateusersession&SessionId=71445D00A301C2F5F12443AC18E4215F&user=jack
```

Please contact Technical Support if you need additional help.

Customizing Function Key Text Recognition

Standards for naming function keys on screens can vary widely, and Presto allows you to identify your own function keys using regular expressions saved in the Presto configuration file config.bws. You can find this file in the root directory of your Presto installation on the IFS. Your Presto installations are under the /esdi/ directory, off the root.



If you intend to make changes to config.bws, make sure you save a backup copy first!

Here is a sample of the function key patterns currently defined:

```
<functionkeypatterns>
<functionkeypattern pattern="[^A-Z=:\/0-9\.:;_\-](P?F([0-9]{1,2})[ ]{0,2}[=:\-][ ]{0,2}
([^ =:0-9#\-]+([ ](P?F[^0-9]{1,2}|[^F >,\*\.#])[^ >,\*\.#]*)*))" fulltextgroup="1"
numbergroup="2" textgroup="3" />
<functionkeypattern pattern="[^A-Z=:\/0-9\.:;_#\-](P?F([0-9]{1,2})[ ]{0,2}[=:\-][ ]{0,2}
([^ =:\-\.0-9]+([ ](P?F[^0-9]{1,2}|[^F >,\*\.#])[^ >,\*\.#]*)*))" fulltextgroup="1"
numbergroup="2" textgroup="3" />
<functionkeypattern pattern="[^A-Z=:\/0-9\.:;_#\-](C([0-9]{1,2})[ ]{0,1}[=:\-][ ]{0,1}
([^ =:\-\.0-9]+([ ](C[^0-9]{1,2}|[^C >,\*\.])[^ >,\*\.]*)*))" fulltextgroup="1"
numbergroup="2" textgroup="3" />
<functionkeypattern pattern="CK([0-9]{1,2})" numbergroup="1" />
<functionkeypattern pattern="[^A-Z=:\/0-9\.:;_#\-](CMD[ /]([0-9]{1,2})[ ]{0,1}[=:\-][ ]
{0,1}(([ ]?(CMD[^ /]|CM[^D ]|C[^M ]|[^C ])[^ ]*)+))" fulltextgroup="1" numbergroup="2"
textgroup="3" />
<functionkeypattern pattern="[^A-Z\/0-9\.:;_#\-]((CMD)?([0-9]{1,2})[ ]{0,1}[\-][ ]{0,1}
([^ =:\-\.0-9]+([ ]?([0-9]{1,2}[^0-9,\.\*\/\-]|[^0-9 ,\.\*\/\-]{2,})[^ ,\.\*\/\-]*)+))[
]" fulltextgroup="1" numbergroup="3" textgroup="4" />
</functionkeypatterns>
```

If you find that Presto isn't consistently identifying your own function keys because you use a different standard than we have anticipated, you can change these regular expressions.



The section "Configuring Your Environment(s)" on page 37 discusses the default options related to function keys and how they are handled.

Contact Technical Support if you need assistance!

Subfile Option Flyout Customizations

This version includes a couple of enhancements to customize the way Presto detects subfile options to build the flyout menu.

Subfile Option Separator

You can now override the separator that Presto looks for when building the flyout option menus. Presto 6.1 includes a JavaScript API for setting/getting the subfile flyout menu separator.

By default, Presto assumes that '=' is the separator. To change this, follow these steps:

1. In the Presto Designer open config.bws (Tools>Advanced Configuration).
2. Find the <detectlastsubfileoption> tag. This tag uses a regular expression to define how options are detected to build the flyout menu. The pattern has three parts:
   - "[a-zA-Z0-9]{1,3}" - The characters before the separator.
   - "=" - The separator.
   - "([^\s\,]+(\s{1,2})?(?![a-zA-Z0-9]{1,3}=))+" - The characters after the separator.
3. Update the separator component. Use [] for multiple values.

For example, if you wish Presto to consider "=" & "-" as separators then your pattern should look like this:

```
[a-zA-Z0-9]{1,3}[=-]([^\s\,]+(\s{1,2})?(?![a-zA-Z0-9]{1,3}=))+
```

At this point Presto will detect patterns with "-" as the separator, for example "3-Test".

The second part of this feature is to update the algorithm that builds the menu, which by default is also expecting an equals sign (=) as the separator. To change this, follow these steps:

1. In the Presto Designer open your site's custom.js file (Tools>User Resources>custom.js).
2. Add this code:

```
TransformSubfileOptions.SetSeparators(['=','-']);
```

Note that pass the function as array with all the supported separators. You can also get a list of the currently used separators with `TransformSubfileOptions.GetSeparators();`

Setting Detection Exceptions

Your application screens or database may include text or field values that are valid in their own context, but happen to resemble subfile options. You can now create a list of values that will be ignored when Presto builds the flyout menus for your screens.

Here is an example of the code you can add:

```
TransformSubfileOptions.SetIgnoredWords(['4=Delete', '2=Change']);
```

You can also retrieve a list of the current ignored words, with this:

```
Getting: TransformSubfileOptions.GetIgnoredWords();
```

Datepicker Auto-Detection

Presto now includes a new feature to recognize date fields and automatically transform them to a datepicker.

This option is enabled by default. To enable the date field auto-detection, open the Presto designer and go to Tools>Advanced Configuration. Find the field named <AutoDatepicker> and set the enabled attribute to '1' (to disable, set to '0'). Then, once you start a new session, any fields that are not already transformed will be checked to determines if they are a date field. If they are determined to be a date field, they are converted to a datepicker.

Presto determines your date fields based on criteria set in the regular expressions <AutoDatepickerPattern> found in the configuration file config.bws (Tools>Advanced Configuration).



If you intend to make changes to these expressions in config.bws, we recommend you save a backup copy first. Unless you are comfortable working with regular expressions, we recommend you contact Technical Support for assistance.

Here is a sample of the date format patterns currently defined:

```
<module name="AutoDatepicker" enabled="1">
<parameter name="DateFormat_ISO_8601">yy-mm-dd</parameter>
      <parameter name="Pattern_ISO_8601">\d{4}-\d{2}-\d{2}</parameter>
      <parameter name="DateFormat_COOKIE">D, dd M yy</parameter>
      <parameter name="Pattern_COOKIE">[a-z]{3}, \d{2} [a-z]{3} \d{4}</parameter>
      <parameter name="DateFormat_RFC_822">D, d M y</parameter>
      <parameter name="Pattern_RFC_822">[a-z]{3}, \d{2} [a-z]{3} \d{2}</parameter>
      <parameter name="DateFormat_RFC_850">DD, dd-M-y</parameter>
      <parameter name="Pattern_RFC_850">[a-z]+, \d{2}-[a-z]{3}-\d{2}</parameter>
// Note - In the real Config.bws there are dozens more date format patterns.
</module>
```

If you find that Presto isn't consistently identifying your own date fields because they use a different format than we have anticipated, you can change or add to these patterns.

In addition to the date-format pattern, there are a couple other options configurable with this feature:

Config Values	Default Value	Possible Values	Description
DatepickerShowOn	both	'both', 'focus', or 'button'	This global option corresponds to the environment-level showOn option found on Date Picker fields. Choose if you want to have the datepicker appear automatically when the field receives focus ('focus'), appear only when a button is clicked ('button'), or when either event takes place ('both').
DatepickerDateFormat		'yy-mm-dd',	This global option corresponds to the environment-level

Config Values	Default Value	Possible Values	Description
		'D, dd M yy', 'D, d M y', 'DD, dd-M-y', 'D, d M y D', 'd M yy D', 'd M y', 'd/m/y', d/m/yy', 'mm/dd/yy', 'mm/dd/y', 'y/m/d', 'y/mm/dd', 'yy/m/d', 'yy/mm/dd', 'd/m/y', 'd/m/yy', 'dd/mm/yy', 'd.m.yy', or 'dd.mm.yy'	dateFormat option found on datepicker fields. Choose the format for parsed and displayed dates.



Some date formats may not be detected even with changes to the regular expression. In these cases, you can still use the Visual Editor or Javascript to transform these fields into a date picker.

Please contact Technical Support if you need assistance.

Presto Launch and URL Parameters

This is a typical Presto run-time URL:

http://192.168.0.1:8022/presto/presto

We have created a number of parameters you can append to this URL for testing and debugging your site, or overriding layouts, skins and languages. Note that most of these are intended for relatively seasoned Presto developers or very specific situations.



You can append multiple parameters to a URL, for example to test a particular skin with a particular language.

Debug

To run Presto in debug mode, append debug=Y to your /prestoadmin/ path:

/prestoadmin/presto?debug=Y

Use this flag to manually set your session to debug mode, typically in preparation for recording a debug log to send to Technical Support.

Replay Log

Use this parameter to replay a recorded debug log manually. Here is an example URL:

/prestoadmin/presto?replay=log/alogfile.log

The value of this parameter (log/alogfile.log) is the path to the debug log relative to your Presto IFS folder.

You can also replay saved debug logs from the debug menu in the Designer.

AutoLogin

You will also see URL parameters used in the context of Auto Logins, as in this example URL:

/presto/presto?new_session=Y&lnknbr=1&StrUpAPI=nexusaccess

Autologins are described later in this chapter, in the section “Creating and Working with Auto Logins” on page 88.

Device

Use a URL parameter to set a session device:

/presto/presto?devicename=mysessname

For further information on this feature, please refer to our online Knowledge Base article 8717.

Clean

Sometimes when debugging an issue with how a screen is rendered, it's useful to see a version of the screen as rendered with NO page-level customizations, just to confirm that the out-of-the-box functionality is OK, and confirm that the issue is related to additional custom changes.

To go to a page with no customizations applied, click the rocket to launch a Presto session in your browser, then append the parameter '?clean=1' to the Presto URL:

presto/presto?clean=1

Then navigate to the page and review the results.



This technique does not disable changes applied through custom.js.

Skin

You can also use a URL parameter to set the site skin at run time:

presto/presto?skin=cascadia

This feature is useful for testing how page enhancements are working with different skins.

Layout

Use the layout parameter to test a page as it would appear in a mobile device:

presto/presto?layout=mobile

This is another useful development tool, as you test the impact of page design changes on various devices.

Config

If you are experimenting with multiple site configuration setting files, you can override to a particular version using this parameter:

presto/presto?config=config2.bws

While this is mainly available to support advanced software testing, it may also be useful in running performance tests with different configuration settings.

Language

Set your site language using the two letter language ID in the URL:

presto/ru/presto

See the section “Configuring Presto for Multi-Language Support” on page 399 for further information.

Tab Order

Use this URL parameter to override the page tab order:

presto/presto?tab=po

This causes your tabbing to behave as it would in the green screen - constrained to the input capable fields on the screen, and not to other elements that your browser might otherwise tab to.

Skip Confirmation

Use this option to set your site to skip the confirmation message that comes up by default when a user attempt to leave (an action which is easy to do in a web browser, but which is akin to

abruptly closing a 5250 session without signing off). Here is an example of the URL:

presto/presto?sc=1

SC stands for "skip confirmation". When this parameter is passed it will not warn you that you are about to leave the page.

Custom Subsystem

Presto now supports the ability to run it's interactive jobs in a custom subsystem.

Please refer to our online Knowledge Base article 15034 for instructions:

```
https://www.myfreschesolutions.com/kb?kbnum=15438
```

Call Technical Support if you have questions!

Access to Page-Level JavaScript customizations in Firefox and Chrome Developer Tools

Allows the user to work with the page level custom JS in FF and Chrome developer tools. The custom JS appears in the browser as PageLevel.js.

Here is an illustration of how you would see this in Chrome:

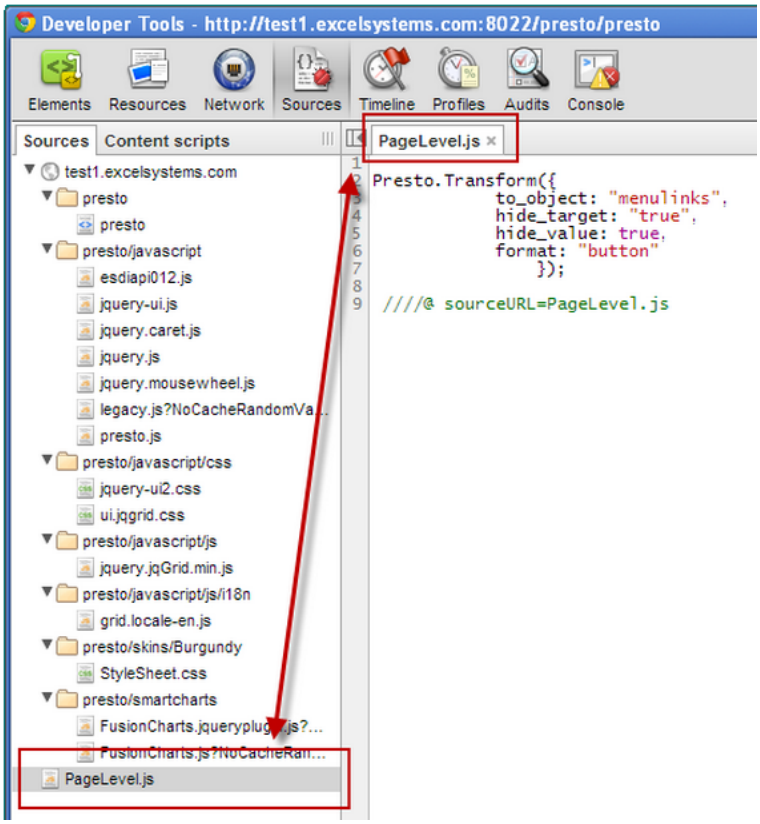


Figure 2-2. Illustration of where you can find the page level JavaScript in Chrome dev tools.

Multiple Data Queues

When a large number of sessions (>1000) are running concurrently in the same environment, it's possible for a single data queue to become locked or full, preventing new sessions. In rare circumstances, such as in a power outage, data queues can become damaged and inaccessible. Using multiple data queues can aid in these situations by allowing new Presto sessions to select a functional queue, skipping the damaged ones, without resulting in server downtime. You can use the option `MaxDqCount`, which allows you to configure multiple data queues for Hub/CGI communication.

To configure the option for multiple data queues, open the Presto designer and go to Tools>Advanced Configuration. Find the following element:

```
<MaxDqCount>0</MaxDqCount>
```

This option is 0 by default. If this option is 0, Presto will use the data queue `PR_CONVQ` as it previously did.

If this option is set to 2 or greater then Presto will randomly pick a `PR_CONVQ##` data queue when the session begins. Note: a session will live in a single data queue, but a single data queue can contain multiple sessions (By default we use `PR_CONVQ` for *all* sessions).

You can run the command `CRTDTAQS`, in your current Presto environment, to create the data queues. First add your current Presto environment (`XL_PRESTO` by default) to your library list, then run the command `CRTDTAQS`.

These are the parameters you need for `CRTDTAQS`:

Parameter	Description
NUMQ	Number of Queues. Must be between 1 and 100.
LIB	Presto Library.
AUTORCL	Automatic Reclaim. *NO, *YES

Example of `CRTDTAQS`:

```
CRTDTAQS NUMQ(2) LIB(XL_PRESTO)
```

This command creates two data queues named `PR_CONVQ00` and `PR_CONVQ01` in the library `XL_PRESTO`.

Bypassing Display Program Messages Screen

There is a new `config.bws` option (Tools>Advanced Configuration) that allows you to bypass the "Display Program Messages" screen when a user signs on to another session.

```
<SkipDisplayProgramMessages>0</SkipDisplayProgramMessages>
```

The value 0 (default) will show the display program messages screen.

The value 1 will bypass and not display the display program messages screen.

The Display Program Messages screen `"Job 222313/USER/DEVICE started on MM/DD/YY at HH:MM:SS in subsystem QINTER Message queue USER is allocated to another job."` is displayed when a user signs on to another session.

This option is useful if you want users to go directly from the sign-on screen to the user's initial program regardless of whether the user has any already active session.

Setting for Non-English Systems

See KB 14896 for instructions on setting up a language file to allow Presto to detect the Display Program Messages screen in your language.

Load Testing with JMeter

To start load testing of your applications running through Presto, we ship a JMeter example test script with the Presto IDE.

See the instructions found in this KB article on myFrescheSolutions KB article 16430

How to create a Custom Error Handler

There is an event called 'PrestoError' triggered when an error is received from the server.

You can bind your own event handler to this event.

The function of your custom error handler should take this form:

```
function CustomPrestoErrorHandler(event, JSONResponse) {}
```

Example of Usage:

```
jQuery(document).off('PrestoError');
jQuery(document).on('PrestoError', function(event, JSONResponse) { alert
(JSONResponse.error); });
```

The format of JSON Respone object is the following:

```
{
       error: "",
       code: "",
       recoverable: true/false,
       critical: true/false
}
```

Example of Errors that trigger Presto Error event:

- If the CGI job cannot communicate with the Hub job.
- If the user manages to submit an invalid value for a given input, the Hub will return an Abnormal Termination error message.
- If communication between the Hub and the Provider fails.
- If the Hub cannot open the config.bws file.

- If there is a version mismatch between the Hub and the IFS.
- If there is an error from the License Server.

Appendix C - Presto.Transform Function Reference

This appendix includes documentation for the client side JavaScript transformation functions used by Presto to either transform existing HTML elements or insert new ones into any Presto page.

The main JavaScript file is presto.js. This is loaded once as part of the HTML framework within which all your Presto pages are presented.

The main object used in presto.js is called Presto.

The Transform, Insert and Select Methods

The two most significant methods in presto.js are Transform and Insert. The Select method is also useful in a more limited set of circumstances.

All three methods accept a JavaScript object, described in JSON notation, as a parameter. The sections below include numerous examples you can reference.



Please visit http://www.json.org/ for further information about 'Jason'[JSON].

The Transform Method

The Transform method operates on an existing HTML element (for example, a text field for a date) and either adds new properties (maxlength, size, cols, rows, etc.) or changes it to a different element (such as a jQuery date picker).

This example changes the HTML element at position 15,15 to a date picker:

```
<script type="text/javascript">
Presto.Transform({
    row: 15,
    col: 15,
    to_object: "datepicker"
});
</script>
```

The Insert Method

Use the Insert method to create new HTML elements and insert them at a specified location on the page.

This example inserts a new element (in this case, an image) at row 10, col 10:

```
<script type="text/javascript">
Presto.Insert({
    row: 10,
    col: 10,
    object: "image",
    src: "images/delete.gif"
});
</script>
```

The Select Method

Use the Select method to select elements on the screen by their row and column. Presto.Select returns the jQuery object with the a list of matched elements.

Here is an example from the tutorial that adds a drop down box to a page:

```
<script type="text/javascript">
Presto.Insert({
    row : "16",
    col : "35",
    object: "select",
    options:
    [
        {"value" : "", "text" : "Choose a Demo Program"},
        {"value" : "CALL PT_MENU", "text" : "PT_MENU"},
        {"value" : "CALL PT_WCUSTS", "text" : "PT_WCUSTS"},
        {"value" : "CALL PT_WITEM", "text" : "PT_WITEM"}
    ],
    onchange: "Presto.Select({row: 20, col: 7}).val(this.value);SendRequest('EN');"
});
</script>
```

When the user selects an option from this list the page submits a request to the server, passing the selected command to be executed.

Presto.Transform Support for Cascading Dropdown Lists

To support cascading dropdown lists (e.g. lists that are loaded when items are selected in higher level lists) Presto.Transform supports a success method. Here is an example:

```
Presto.Transform(
{
    row: 5,
    col: 19,
    to_object: 'select',
    sql_options:
    {
        name: "cust_DD",
        params:
        {
            cust_number: "selector: [row='5'][col='19']"
        }
    },
    success: function() { //do more stuff }
});
```

JavaScript Objects Used for the Transform and Insert Methods

The object used for the methods listed above have two different broad categories of properties:

1. **The Selector.** The selector can any combination of a single row/column pair, or a range of rows, an array of rows, a range of columns or an array of columns. Or another way a selector can be defined is as a regular jQuery selector.
2. **The Set of Transformation Properties.** These can be properties that add characteristics to an existing HTML element, such as css style, text or background color, or they can be properties that change the type of HTML element. For example, you can use a to_object property to change the HTML element from a text field to a drop down box.

The following sections discuss these.

The Selector Properties

The Selector properties are row and col (column). If you specify row, you must also specify col, and vice versa.

Selector	Description
row	single value, array of values, or range, specified using this object: { first : "x", last: "y" }
col	single value, array of values, or range, specified using this object: { first : "x", last: "y" }

Here are some examples of specifying properties to select one or more elements on a page for transformations:

Change a style of an object in col 2 and row 2:

```
{
    row: 2,
    col: 2,
    css: {border: "red 1px dashed", "font-size": "18px"}
}
```

Change all values in rows 1 - 4, column 2:

```
{
    row: {first: 1, last: 4 },
    col: 2,
    css: {border: "red 1px dashed","font-size": "18px"  }
}
```

Change all cols from col 2 to col 21, rows from 1 to 4:

```
{
    row:    {   first: 1,   last: 4 },
    col:    {   first: 2,       last: 21    },
    css: {border: "red 1px dashed", "font-size": "18px"}
}
```

Change all values in rows 1, 3, 5, column 2:

```
{
```

```
    row: ["1", "3", "5"],
    col: "2",
    css: {"border": "red 1px dashed", "font-size": "18px"}
}
```

Change all values in rows 1, 3, 5, columns 2, 21, 70:

```
{
    row: ["1", "3", "5"],
    col: ["2", "21", "70"],
    css: {"border": "red 1px dashed", "font-size": "18px"}
}
```

Change all values in rows 1-4, columns 2, 21, 70:

```
{
    row: {first: "1", last: "4"},
    col: ["2", "21", "70"],
    css: {"border": "red 1px dashed", "font-size": "18px"}
}
```

Transformation Properties

The following is a list of properties for the object supplied as a parameter to the Transform and Insert methods that are used to either change an existing HTML element or insert a new one.

to_object, object

You can use **to_object** to change an HTML element(s) from one type of object to another or to insert a new one. The element(s) are identified using either the selector, or row/col properties, as described above.

For example, in most Presto pages the standard input control is a text field. Use this property to change the text field to some other HTML element that accepts input, such as radio buttons or drop down boxes.

Usage

Specify this property when you want to change an existing HTML from one type to another. For example, if the HTML element is currently a text field and you want it to be a text area, you will need to specify **to_object: "textarea"**. However, if you want it to remain a text field, you can safely omit this property. If you want to add new text area, then you have to use **Insert** method instead and property **object: "textarea"**.

Possible values:

Here are the possible values:

- **"datepicker"** - Adds date picker to a target. You should usually choose an input text box element to apply this to. This has a clickable calendar image that allows the user to select a date from the calendar and return it in a specified format to a text field. To customize it you can specify more parameters supported by date picker, for more info see [http://docs.jquery.com/UI/Datepicker#options jQuery Datepicker options]
- **"label"** - Changes the target to a label object.
- **"textarea"** - Changes the target to a text area with cols="100" and rows="5".
- **"button"** - Changes the element to type button.
- **"image"** - Changes the target to an image.
- **"input"** - Changes the target to a text field. You can also use this value to add more properties to an existing text field.
- **"select"** - Changes the target to a drop down box. This is an object of key/value pairs where the "key" is used as a value which will be submitted to the server and "value" is the visible label for this "key".
- **"radios"** - Changes the target to one or more radio buttons. This is an object of key/value pairs where the "key" is used as a value which will be submitted to the server and "value" is the visible label for this "key".
- **"iframe"** - Creates an iframe and navigates it to the URL specified in the src parameter.
- **"a"** - Transforms the selected object into an anchor (hyperlink).
- **"clickableoptions"** - Transforms the selected element(s) into clickable images. Supports the property 'keep_target', with possible values of true & false.

- **"menulinks"** - Transforms the selected element(s) into clickable links. There are a number of other possible options with this transformation as well.
- **"checkboxes"** - Changes the target to one or more check boxes. This is an object of key/value pairs where the "key" is used as a value which will be submitted to the server and "value" is the visible label for this "key".
- **"autocomplete"** - Adds autocomplete functionality to an input-capable field.
- **"export data"** - Creates an element (either a link, button or image) that when clicked triggers the download of a data set.

The next section includes some examples.

Examples

Here are some examples:

Create two radio buttons by transforming existing element, with text placed to the right of each button:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "radios",
    value: "<%=screen(12,12,1)%>",
    options:
    [
        {"value" : "Y", "text" : "Yes"},
        {"value" : "N", "text" : "No"}
    ],
    text_pos: "after"
});
```

Insert a default image if a specified image doesn't exist.

```
<script>
  Presto.Transform({
    row: 11,
    col: 50,
    object: "image",
    src: "<%screen(10,4,25)%>",
    onerror: "Presto.Transform({row: 11, col: 50, object: 'image', src:
'images/notfound.jpg'})"});
</script>
```

Replace a field with a radio button for the two available options:

```
<script type="text/javascript">
Presto.Transform({
    row: 18,
    col: 29,
    value: "<%=screen(18,29,15)%>",
    to_object: "radios",
    options:
    [
        {"value" : "1", "text" : "CASH"},
        {"value" : "2", "text" : "ACCOUNT"}
    ],
    text_pos: "after"
});
</script>
```

The database file has either 1 or 2 in that field, but to the user this displays CASH or ACCOUNT.

Create a check box with text placed to the left:

```
Presto.Insert({
    row: 12,
    col: 12,
    object: "checkboxes",
    options:
    [
        {"value" : "Y", "text" : "Yes"},
        {"value" : "N", "text" : "No"}
    ],
    text_pos: "before",
});
```

Create two check boxes by adding new elements before existing one, with text placed to the right of each check box and absolute position them:

```
Presto.Insert({
    row: 12,
    col: 12,
    object: "checkboxes",
    options:
    [
        {"value" : "Y", "text" : "Yes"},
        {"value" : "N", "text" : "No"}
    ],
    text_pos: "after",
    css: {
        position: "absolute",
        left: "10px",
        top: "20px"
    }
});
```

Change text color to red, background to yellow:

```
Presto.Transform({
    row: 12,
    col: 12,
    css: {color: "red", "background-color": "yellow"}
});
```

Change to a date picker:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "datepicker"
});
```

Change to an image:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "image",
    src: "images/digby.jpg"
});
```

Change to a text field:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "input",
    type: "text"
});
```

Change to a text field with a size of 100, maximum number of characters inputtable of 120, and make the border of box red, and the text red:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "input",
    size: "100",
    maxlength: "120",
    css: {border: "3px solid red", color: "red"}
});
```

Change to a drop down box with two selectable values: "Yes" returns "Y", "No" returns "N" and "Yes" selected by default:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "select",
    value: "Y",
    options:
    [
        {"value" : "Y", "text" : "Yes"},
        {"value" : "N", "text" : "No"}
    ],
});
```

Same as last example, except also style the box and, if an item in the box is selected that is different from the current value, act as if the user pressed the Enter key:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "select",
    value: "Y",
    options:
    [
        {"value" : "Y", "text" : "Yes"},
        {"value" : "N", "text" : "No"}
    ],
    css: {border: "2px dashed red"},
    onchange: "alert('Value is changed!')"
});
```

Change a single element to a drop down box. Use screen text to derive both the value and text associated with it:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "select",
    options: "[2=Change] [4=Delete] [5=Display]"
});
```

Same as previous example, except get the text dynamically from the screen, on row 7, column 2 for 79 characters:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "select",
    options: "<%=screen(7,2,79)%>"
});
```

Change single element to checkboxes with three options:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "checkboxes",
    options:
    [
        {"value" : "R", "text" : "Red"},
        {"value" : "G", "text" : "Green"},
        {"value" : "B", "text" : "Blue"}
    ],
    text_pos: "after"
});
```

Change single element to checkboxes and absolute position it:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "checkboxes",
    options:
    [
        {"value" : "R", "text" : "Red"},
        {"value" : "G", "text" : "Green"},
        {"value" : "B", "text" : "Blue"}
    ],
    text_pos: "after",
    css : {
        position: "absolute",
        left: "10px",
        top: "20px"
    }
});
```

prefix, suffix

You can add prefix or suffix to any transformation by using **prefix** and **suffix** properties.

Usage

Use these as follows:

- **prefix** - Inject portion of Text or HTML before selected element
- **suffix** - Inject portion of Text or HTML after selected element

Usually **prefix** and **suffix** should be used together to create a container.

Example

Here is an example:

Insert new text area before element at row 12, column 12 and set text area value, prefix and suffix:

```
Presto.Insert({
    row: 12,
    col: 12,
    object: "textarea",
    onclick: "alert('Thank you for clicking!')",
    value: "INSERT EXAMPLE",
    insert_pos: "before",
    rows: 5, cols: 30,
    prefix: "<div>PREFIX</div>",
    suffix: "<div>SUFFIX</div>"
});
```

source

This is required property for the autocomplete transformation, and used in others as well.

Example Without a Dynamic Search Term

Here is an example without a dynamic search term:

```
select distinct imcat From xl_prdemo/mu_itmf
```

The query name is 'Inventory'. The transformation would look like this:

```
Presto.Transform(   {
    row: 12,
    col: 53,
    to_object: "autocomplete",
    source: {
        name: "inventory"
    }
});
```

Example Using a Dynamic Search Term

Here is another example with a dynamic search term:

```
select distinct imcat from xl_prdemo/mu_itmf where lower(imcat) like :term
```

The query name is 'Inventory2', and this is the Presto transformation:

```
Presto.Transform({
    row: 11,
    col: 53,
    to_object: "autocomplete",
    source: {
        name: "inventory2",
        param: "term"
    }
});
```

Static List Example

Here is an example generating a static list:

```
Presto.Transform({
    row: 10,
    col: 53,
    to_object: "autocomplete",
    source: ["Amps", "Guitars", "Ukulele" ]
});
```

value

You can use **value** to assign a value to any transformation object.

Here are some examples:

Insert text area with value:

```
Presto.Insert({
    row: 12,
    col: 12,
    object: "textarea",
    value: "INSERT EXAMPLE",
    insert_pos: "before",
    rows: 5, cols: 30
});
```

Transform to drop down boxes and select some value:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "select",
    options:
    [
        {"value" : "Y", "text" : "Yes"},
        {"value" : "N", "text" : "No"}
    ],
    value: "N"
});
```

Any Valid HTML Attribute

This feature lets you use any valid HTML attribute for the target HTML element, whether it is one you are transforming with the Transform method or inserting into the page with the Insert method.

For example, if the HTML element is a text field, you could specify maxlength : "20" to assign the HTML attribute maxlength to the element. Or, for a text area, you could specify rows: "8" , cols: "250" to change/create a text area with 8 rows and 250 columns.

Usage

Available for types with values of "input", "image", "textarea", "input", "select", "radios" or "checkboxes"

Possible Values

Any valid HTML attribute for the appropriate HTML element.

Examples

Here are some examples:

Define height and width for an image transformation:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "image",
    src: "images/delete.gif",
```

```
    height: "15px",
    width: "15px"
});
```

src

Specifies the path and file name of an image file.

Usage

Available for "image" transformation.

Example

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "image",
    src: "images/delete.gif",
    height: "15px",
    width: "15px"
});
```

css

You can specify any stylesheet selectors for this property, in order to add custom CSS to an element.

Example

Here is an example:

```
Presto.Insert({
    row: 12,
    col: 12,
    object: "checkboxes",
    options:
    [
        {"value" : "Y", "text" : "Yes"},
        {"value" : "N", "text" : "No"}
    ],
    text_pos: "after",
    insert_pos: "before",
    css:
    {
        position: "absolute",
        left: "10px",
        top: "20px"
    }
});
```

options

This property is only valid when the type property is radios (radio buttons), checkboxes (check boxes) or select (drop down box).

Its value must be an object, where attribute name represents value of an option and value is text (title) of a given option, for example: {"Y" : "Yes", "N" : "No"}. These represent the option value assigned to the form HTML element and the associated text shown to the user for that option,

respectively. In the case of radio buttons and check boxes, the text may appear after (to the right) or before (the left) of the radio button or check box, depending on the value of the **text_pos** property. In the case of drop down boxes, the text appears in list shown in the drop down.

Usage

Available for radios, checkboxes and select transformations, as well as the custom clickableoptions transformation shipped with Presto.

Example

Here is an example:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "select",
    options:
    [
        {"value" : "Y", "text" : "Yes"},
        {"value" : "N", "text" : "No"}
    ]
});
```

text_pos

Determines location of text in relation to each radio button in a group of buttons or to each check box.

Usage

Only valid for **radios** and **checkboxes**.

Possible Values

Here are the possible values:

- **after** or **right** - text appears after (to right of) a button
- **before** or **left** - text appears before (to left of) a button.

Example

Here is an example:

```
Presto.Transform({
    row: 12,
    col: 12,
    to_object: "radios",
    options:
    [
        {"value" : "Y", "text" : "Yes"},
        {"value" : "N", "text" : "No"}
    ],
    text_pos: "after"
});
```

Events Property

You can bind events by specifying event name like: onclick, onchange and action which should be performed on this event.

Events can be specified in property **events** or without that property. Event's logic can be specified as a function or a string.

Example

Here is an example:

```
Presto.Transform({
    row: 12,
    col: 12,
    events:
    {
        onclick: function()
        {
            alert('Test!');
        }
    }
});

Presto.Transform({
    row: 12,
    col: 12,
    onclick: "alert('Test!');"
});
```

Presto Autocomplete Transformation Properties

Presto's autocomplete transformation is one of the most powerful and useful. Below is a description of this transformation's properties.

The autocomplete transformation transforms selected element(s) to input fields using jQuery.autocomplete.

The only required parameter it expects is called source which can of four different types:

1. An array with local data, which can look like: ["Africa", "Europe", "America"]
2. A string which specifies a URL with the data.
3. A callback function that returns data for autocomplete.

For more information about these three types please refer to the jQuery.autocomplete documentation online.

The fourth alternative is an object which is considered as a request to execute stored SQL query in order to obtain a data. This is the approach taken in Presto.

In this case the following parameters are expected:

- **name** - The name of the query.
- **param** - An optional name of the parameter that should be used for a search term. If it is not provided then plugin will search for a term in the entire result set.
- **params** - This is optional. Hard coded values for the parameters expected by the query.

For additional information on the source parameter and examples, please refer to the section "source" on page 430.

Presto menulinks Transformation Properties

Presto's menulinks transformation has a number of unique properties. These are described below.

target

Use this property to specify the target field for a menu converted to clickable links with the menulinks transformation.

Usage

Use this property with menulinks transformations. In most cases you don't need to specify this parameter, as the menu option is returned to the default input capable field on the screen. However, if the page has more than one input capable field, you need to specify the field to return the menu option to.

Example

Here is an illustration of this property:

```
<script type="text/javascript">Presto.Transform({
    to_object: "menulinks",
    target: {row: 20, col: 5}
});
</script>
```

This transforms the green screen menu on this page, returning the menu option for the selected value to the field at row 20, column 5.

hide_value

Use the optional hide_value property to hide the menu option numbers or literals. This option defaults to false.

Usage

You can set this property to true if you want your users to only use their mouse to click the menu text hyperlinks, rather than entering the options by hand.

Example

Here is an example of this property:

```
<script type="text/javascript">
Presto.Transform({
    to_object: "menulinks",
    hide_value: false,
    hide_target: false
});
</script>
```

This examples leaves both the menu options and the menu option input field visible, allowing the user to read the options and enter them manually if they wish.

Transforming your menu like this:

```
<script type="text/javascript">
```

```
Presto.Transform({
    to_object: "menulinks",
    hide_value: true,
    hide_target: true
});
</script>
```

would require that user click the option text hyperlinks.

hide_target

Use the optional hide_target property to hide the menu option field. This option defaults to false.

Usage

If your menu includes just a few options and an option field, you may want to make the page look more like a normal web page by hiding the options field. If you want to allow users to still enter menu values with their keyboard and the Enter key, you can leave the option field visible.

If you have a menu with a command line as the option field, you will probably want to leave this visible (the default).

Example

Here is an example of this property:

```
<script type="text/javascript">
Presto.Transform({
    to_object: "menulinks",
    hide_target: false
});
</script>
```

This examples leaves both the menu options and the menu option input field visible, allowing the user to read the options and enter them manually if they wish.

Transforming your menu like this:

```
<script type="text/javascript">
Presto.Transform({
    to_object: "menulinks",
    hide_value: true,
    hide_target: true
});
</script>
```

would require that user click the option text hyperlinks.

pattern_label, pattern_value, pattern_label_group and pattern_value_group

These additional options allow you to use regular expressions to allow you to use the menulinks transformations with menus where the options or labels have non-standard formats.

Contact Technical Support for assistance with these options.

format

The menulinks transformation by default converts your menu options to hyperlinks, but you can also override the format to convert them to buttons.

Here is an example:

```
<script type="text/javascript">
Presto.Transform({
    to_object: "menulinks",
    hide_target: true,
    hide_value: true,
    format: "button"
    });
</script>
```

Customizing MenuLinks

It's possible to customize menulinks via code in custom.js. For example, if you want to hide the menu numbers by default then adding the following code to custom.js does the trick:

```
jQuery(document).unbind("TransformMenuLinks");
jQuery(document).bind("TransformMenuLinks", function() {
   Presto.Transform({to_object: "menulinks", hide_value: true});
});
```

Contact technical support if you have questions!

Presto clickableoptions Transformation Properties

The clickableoptions transformation is also a unique feature in Presto that has some unique options.

Here is an example of this transformation, which replaces a set of subfile options with clickable image links:

```
<script type="text/javascript">
Presto.Transform({
    row: {first: 12, last: 21},
    col: 3,
    to_object: "clickableoptions",
    keep_target: true,
    options: {
"2=Change" : "images/edit.gif",
"4=Delete" : "images/delete.gif",
"5=Display" : "images/view.gif",
"7=Orders" : "images/details.gif"
        }
    });
</script>
```

This example is illustrated in the section "Replacing Subfile Options with Clickable Links" in the online tutorial.

keep_target

This option allows you to set whether the transformation should keep the input box visible or not. The default is 'false', to replace the

Presto fileupload Transformation Properties

Presto's fileupload transformation allows users to upload files to IFS.

The object type is called fileupload.

It will upload the file to the IFS in the folder: `/esdi/xl_presto/uploads` where `xl_presto` is the name of your Presto environment.

Here is an example that could be placed in Page-Level Javascript to insert a button to upload a file:

```
jQuery(document).one("PageComplete", function()
{
    Presto.Insert({
"object": "fileupload",
"value": "Upload File",
"vrow": "10",
"vcol": "20",
"events":
        {
"done": function() {alert('File uploaded successfully.');},
"fail": function() {alert('File upload error.');}
        }

    });
});
```

Note: for security reasons you must specify which types of documents Presto will allow to be uploaded to the server. Follow Knowedge Base article 16384 for instructions.

Presto Flyout Transformation Properties

The Presto Flyout transformation allows transforming an input on the screen into a Flyout menu.

Here are a few examples:

```
Presto.Transform({
    id: 'f0',
    to_object: 'flyout',
    options:
    [
        {"value" : "a", "text" : "ALPHA"},
        {"value" : "b", "text" : "BETA"},
        {"value" : "d", "text" : "DELTA"},
        {"value" : "e", "text" : "EPSILON"},
        {"value" : "g", "text" : "GAMMA"}
    ],
    submit: true,
    show_value: true,
    timeout: 200
});

Presto.Insert({
    row: 15,
    col: 20,
    object: 'flyout',
    options:
    [
        {"value" : "Y", "text" : "YES"},
        {"value" : "N", "text" : "NO"}
    ],
    show_value: false
});

Presto.Transform({
    id: 'f3',
    to_object: 'flyout',
    sql_options: {
        name: 'mytest'
    }
});
```

JavaScript Functions to Retrieve and Set Presto Variables

Presto allows you to interact with a number of the variables in presto.js. If you are looking at a variable that doesn't seem designed to be worked with, please contact our technical teams to discuss the best strategy.

Getting and Setting Variable Information

You can use a pair of Javascript function Presto.GetScreenProperty and Presto.SetScreenProperty to retrieve and set variable information:

Here is an example:

```
Presto.SetScreenProperty('selectedrow', 10);

var bIsWindow = Presto.GetScreenProperty('iswindow');
```

The supported parameters are:

- FieldCount
- IsJsonScreen
- isOaScreen
- IsSignonScreen
- IsWindow
- Pagekey
- PageSeq
- PageSize
- SelectedCol/SelectCol
- SelectedRow/SelectRow
- HasTransform (New in 8.6)
- HasJavascript (New in 8.6)
- HasCustomHtml (New in 8.6)

Getting and Setting Environment Values

Note: This functionality was available since 2013. In v7.1 improved functions were introducred with new names.

- Before 7.1: SetEnvironmentSetting() and GetStateVariable()
- 7.1+: SetEnvironmentSetting() and GetEnvironmentSetting()
- Backwards compatible. ie, GetStateVariable() still works.

You can set environment-settings values as follows:

```
Presto.SetEnvironmentSetting(sSetting, sValue);
```

Here is an example:

```
Presto.SetEnvironmentSetting("SessionEndResponse", "Session ends!");
```

Supported settings:

- "AdvanceCursor"
- "DisableTabProcessing"
- "ErrorHandlerType" sets the string value passed for the error handling type ("default", "dialog", or "ignore").
- "NumericOnlySign"
- "Scrolling"
- "SessionEndResponse" sets the string value passed as the message displayed on screen when user disconnects from Presto.
- "SessionReloadResponse" sets the string value passed as the message displayed on screen when user reload session from Presto.
- "SpoolfileDefaults" allow users to pass an object with properties defining default filter criteria for spool-file. Please visit the following Knowledge Base article 10401 article for all valid options: https://www.myfreschesolutions.com/kb?kbnum=10401
- "TypeOver" sets true or false, whether the typing should add or overwrite the text after the cursor.
- "UsePositionalOrdering"

The following settings have been moved to Presto.GetScreenProperty and Presto.SetScreenProperty (see above). They are still supported but deprecated for use with Presto.SetEnvironmentSetting and Presto.GetEnvironmentSetting.

- FieldCount
- IsJsonScreen
- isOaScreen
- IsSignonScreen
- IsWindow
- Pagekey
- PageSeq
- PageSize
- SelectedCol/SelectCol
- SelectedRow/SelectRow

Appendix D - Presto.ExecuteSQL

Use the Presto.ExecuteSQL method to integrate results of SQL queries into custom JavaScript applications. It references SQL queries that you have created via the Presto Designer and are saved on your server.

Technical Details

Presto.ExecuteSql expects an object as a parameter. Typical usage will look like:

```
Presto.ExecuteSql({
      format: "json",
      dataType: "json",
      name: "query_name",
      cache: true,
      params:
      {
              param1 : "value1",
              param2 : "value2"
      },
      success: function(data, textStatus, XMLHttpRequest)
      {
              // do something with the data
      },
      error: function(XMLHttpRequest, textStatus, errorThrown)
      {
      },
      complete: function(XMLHttpRequest, textStatus)
      {
      }
});
```

The parameters are described below.

Parameters

Presto.ExecuteSql expects an object as the only input parameter. The object should contain the name of a query to execute, the desired format of the output and references to the functions that will be executed in case of success or error. You can also provide a dataType to force the result object to be in the desired format.

Depending on the outcome, the “success” or “error” handler is triggered.

format
Use the 'format' to specify which template to use. The shipped templates are in the "table_ templates" folder of your Presto IFS directory (esdi/presto_lib). The default value is 'html'.

Currently four templates (formats) are supported: html, json, json_array and json_list. If the format is not specified, the "html" format is used.

The json_array template is used internally for drop down boxes and json_list is used for autocompletes.

name
This is the name of the query you have created via the Presto Designer (Tools>SQL Queries).

params
This is the list of parameters that the requested query expects. See the example above for guidance on the format of these.



The options dataType, success and error are parameters of jquery.ajax. Please refer to the documentation at http://api.jquery.com/jQuery.ajax/ for additional details.

---

dataType
The dataType is optional, but when added, specifies the type of data you're expecting back from the server. If the datatype is not specified, it will be detected automatically.

success(data, textStatus, jqXHR)
The success parameter identifies a function to be called if the request succeeds. This is optional.

error(jqXHR, textStatus, errorThrown)
The error parameter identifies a function to be called if the request fails. This is also optional.

session
This is an optional session ID on which to execute the SQL.

cache
This is an optional variable to tell Presto to cache the results of the query.

The following will clear the cache:

```
CSqlCache.GetInstance().Clear()
```

complete
The complete parameter identifies a function to be called once the request has run and is complete. It will be called after the "success" or "error" function.

Presto.OpenSession & Presto.CloseSession

You can use Presto.OpenSession and Presto.CloseSession to create standalone, SQL enabled JavaScript applications. This opens the door to many possibilities.

Presto.OpenSession expects the following parameters wrapped into object:

Parameter	Description
user	Username to login to the IBM server.
password	password of the user
success	method which will receive newly created Session ID as its parameter
error	method which will handle error

Presto.CloseSession expects the following parameters wrapped into object:

Parameter	Description
success	Method which will handle successfully closed session and gets message returned by the server.
error	Method which will handle error.
session	Session ID you want to close.

Here is a code sample:

```
Presto.OpenSession({
    user: "YOUR_USERNAME",
    password: "******",
    success: HandleOpenSessionSuccess,
    error: function(sErrorMessage)
    {
        // handle error
    }
});

function HandleOpenSessionSuccess(sSessionId)
{
    Presto.ExecuteSql({
       format: "html",
       dataType: "html",
       name: "test",
       success: function(data, textStatus, XMLHttpRequest)
       {
           // do something with the data
           Presto.CloseSession({success: function(s) {alert("Session has been closed
" + s)}, session: sSessionId});
       },
       error: function(XMLHttpRequest, textStatus, errorThrown)
       {
           // handle SQL error
       },
       // pass session ID
       session: sSessionId
    });
```

```
}
```

Please check our online Knowledge Base for examples, or call Technical Support if you have questions!

Appendix E - Presto JavaScript API

Presto contains a JavaScript API that can be used outside of a standard Presto environment to create fully customized Web Applications around 5250 screens.

An instance of PrestoAPI is created using the "new" keyword:

```
var prestoApi = new PrestoAPI();
```

The following is a list of all public API functions as well as their purpose.



Presto does not currently support macros when you are using OA. In addition, a non-OA program must be your first entry point of the API call.

Instance Methods

Constructor

Description

Used to create a new instance of the PrestoAPI. Takes an object parameter with API options. Current available options:

prestoPath - The URL path to the “presto.pgm” object (Default: /presto/ presto).

Method Prototype

PrestoAPI (pathObject)

Parameter Descriptions

pathObject	The URL path to the `'presto.pgm'` object.

Example

Here is an example of how to use this method:

```
var prestoApi = new PrestoAPI({
    prestoPath: '/presto/presto'
});
```

AutoLogin

Description

Utilize the specified Presto AutoLogin to start a new session. The first screen of the Autologin will be passed to the callback function.

Method Prototype

AutoLogin(linkNumber, callback)

Parameter Descriptions

linkNumber	Link number. Part of the URL.
callback	Callback function passed the first screen data of the autologin.

Example

Here is an example of how to use this method:

```
prestoApi.AutoLogin(1, function(oData) {
    console.log(oData);
});
```

OpenSession

Description

Start a new Presto session to open at the users normal login program. The data from the first screen after the login/messages screen will be passed to the callback function.

Method Prototype

OpenSession(optionsObject, callback)

Parameter Descriptions

optionsObject	An object containing options. Currently Available options are: **user** – The username to log in with. **password** – The password to log in with. **libl** – The library list to set for this session (Space-separated list of libraries). **program** – The initial program to call. **l ibrary** – The library for the initial program to call. **menu** – The initial menu to call. **debug** – Whether to turn debug mode on (true/false).
callback	Callback function with data passed from first login/messages screen.

Example
Here is an example of how to use this method:

```
prestoApi.OpenSession({
    user: 'TESTER',
    password: 'password',
    libl: 'XL_PR69 XL_WEBSPT',
    program: 'prexec',
    library: 'XL_PR69'
}, function (oData) {
    console.log(oData);
});
```

EndSession

Description
End the Presto session, closing the interactive job and freeing allocated resources.

Method Prototype
EndSession()

Example
Here is an example of how to use this method:

```
prestoApi.EndSession();
```

SubmitRequest

Description
Send the current screen data to the server. The resulting screen data is passed to the callback function.

Method Prototype
SubmitRequest(command, callback)

Parameter Descriptions

command	A command string.
callback	Callback function passed the resulting screen data.

Example
Here is an example of how to use this method:

```
prestoApi.SubmitRequest('EN', function(oData) {
console.log(oData);
});
```

PlayMacro

Description
Play the specified macro on the server. The 'macro' parameter can be an array of macro commands, a pre-defined macro name as a string, or a dynamically generated macro string.

Depending on the macro commands, the callback function will be passed back either the last screen visited, or the screen for which GetScreen() is called within the macro.

Method Prototype

PlayMacro(macro, callback)

Parameter Descriptions

macro	This parameter can be the following: • An array of Macro commands. • A string consisting of the pre-defined macro name. • A dynamically generated macro string.
callback	Callback function. This function can be passed either the data of the last screen visited or the screen for which GetScreen() is called within the macro.

Example

Here is an example of how to use this method:

```
prestoApi.PlayMacro([
'Write(4,34,"PT_WCUSTS")',
    'SendCommand("EN")',
    'GetScreen()',
    'SendCommand("03")'
], function(oData) {
    console.log(oData);
});
```

Window-Specific Methods

The methods that allow you to modify Windows within Presto are described in the following KBs:

Setting and Getting the Window Title

Get Window Title

To get the title of the window, you can use the Presto.Window.GetTitle() function.

```
var winTitle = Presto.Window.GetTitle();
```

Set Window Title

To set the title of the window, you can use the Presto.Window.SetTitle(newTitle) function. The newTitle argument is a string that will become the new title of the window:

```
var newWinTitle = "New Title!";
Presto.Window.SetTitle(newWinTitle); // Sets the window title to "New Title!"
```

Setting and Getting the Window Position

Get Window Position

To get the position of the window, you can use the Presto.Window.GetPosition() function.

```
var winPosition = Presto.Window.GetPosition();
```

Set Window Position

To set the position of the window, you can use the Presto.Window.SetPosition(position) function. The position argument is an object with row and col properties

```
var newWinPosition = {row: 2, col: 2};
Presto.Window.SetPosition(newWinPosition); // Sets the window position to row 2,
column 2
```

Setting and Getting the Window Size

Get Window Size

To get the size of the window, you can use the Presto.Window.GetSize() function.

```
var winSize = Presto.Window.GetSize();
```

Set Window Size

To set the size of the window, you can use the Presto.Window.SetSize(size) function. The size argument is an object with rows and cols properties.

```
var newWinSize = {rows: 20, cols: 20};
Presto.Window.SetSize(newWinSize); // Sets the window size to 20 rows and 20 columns
```

See also this KBKB 14959

Modifying and Working with Windows

The Presto Window API allows you to modify and interact with the active window within the Presto environment. This includes getting and setting the window's title, position, and size.

Window Title

Presto.Window.GetTitle()

This method allows you to retrieve the title text of the active window.

Returns: a string representing the title of the active window.

```
var winTitle = Presto.Window.GetTitle();
```

Presto.Window.SetTitle(sNewTitle)

This method allows you to set the title of the active window.

Parameter: sNewTitle - a string representing the new title for the active window.

```
var newWinTitle = "New Window Title";
```

Window Position

Presto.Window.GetPosition()

This method returns an object representing the row and column of the active window.

Returns: an object with properties row and col.

```
var winPosition = Presto.Window.GetPosition();
```

Presto.Window.SetPosition(NewPosition)

This method allows you to set the position of the active window. It accepts decimal values for finer control.

Parameter: NewPosition - an object with properties row and col.

```
var newWinPosition = {row: 3.5, col: 7.2};
Presto.Window.SetPosition(newWinPosition); // Sets the window position
```

Window Size

Presto.Window.GetSize()

This method returns an object representing the size of the active window.

Returns: an object with properties rows and cols.

```
var winSize = Presto.Window.GetSize();
```

Presto.Window.SetSize(NewSize)

This method allows you to set the size of the active window. It accepts decimal values for finer control.

Parameter: NewSize - an object with properties rows and cols.

```
var newWinSize = {rows: 18.5, cols: 24.7};
Presto.Window.SetSize(newWinSize); // Sets the window size
```

See also this KB: KB 14920

OA-Specific Methods

SetFieldValue

Description
Set the specified OA field's value. Value is only sent to the server when SubmitRequest is called.

Method Prototype
SetFieldValue(fieldName, value)

Parameter Descriptions

fieldName	The name of the OA field.
value	The value to set the OA field.

Example
Here is an example of how to use this method:

```
prestoApi.SetFieldValue('@MNUOPT', 'DEMO_PRX');
```

SetSubfileFieldValue

Description
Set the specified OA field's value. Value is only sent to the server when SubmitRequest is called.

Method Prototype
SetSubfileFieldValue(fieldName, rowNumber, Value)

Parameter Descriptions

fieldName	The name of the OA subfile field.
rowNumber	The row of the OA subfile field.
value	Value to set the OA subfile field.

Example
Here is an example of how to use this method:

```
prestoApi.SetSubfieldFieldValue('CMCUST', 5, '1');
```

Static Methods

GetConstant

Description
Return an object containing data for the constant at the given row and column in the supplied screen data. Returns an empty object if a constant is not found.

Method Prototype
Object GetConstant(screenData, row, col)

Return Value Description

Object	Return an object containing data for the constant at the given row and column in the supplied screen data. Returns an empty object if a constant is not found.

Parameter Descriptions

screenData	Screen data.
row	The row of the screen constant in the supplied screen data.
col	The column of the screen constant in the supplied screen data..

Example

Here is an example of how to use this method:

```
var sVal = PrestoAPI.GetConstant(oData, 12, 7).value;
```

GetField

Description

Return an object containing data for the field at the given row and column in the supplied screen data. Returns an empty object if a field is not found.

Method Prototype

Object GetField(screenData, row, col)

Return Value Description

Object	Return an object containing data for the field at the given row and column in the supplied screen data. Returns an empty object if a field is not found.

Parameter Descriptions

screenData	Screen data.
row	Row of the field in the supplied screen data.
col	Column of the field in the supplied screen data.

Example

Here is an example of how to use this method:

```
var sVal = PrestoAPI.GetField(oData, 12, 7).value;
```

Appendix F - Insights Integration

Using the Insights tool, Presto now helps site administrators to visualize the user behavior in their applications. This includes getting an overview of the screens that users are accessing most often, allowing you to spend time customizing what is used instead of trying to customize every screen in their programs. Then you can plan which parts of the application to modernize first, save time, and deliver better business value:

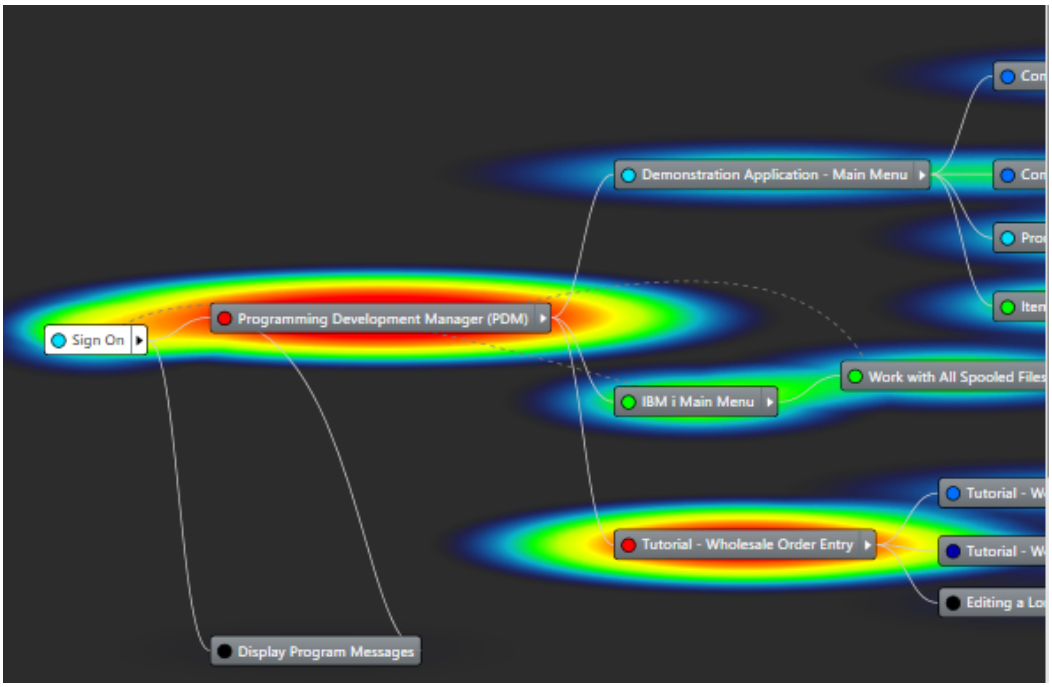


Figure 2-1. Heatmap Graph showing the most visited screens.

Installation

The Insights tool is installed alongside the Presto IDE starting with version 8.1. It will be installed to the "Insights" folder inside the Presto installation folder.

Recordings

To visualize user behavior, you'll first need to record it. The Insights tool depends on .nl recording files. As of Presto 8.1, you can use the recording capabilities in Presto to collect data

on individual users as they access screens in the application, creating individual .nl recording files for each user session.

To enable the creation of .nl recording files, you can either enable "<insight>Y</insight>" in config.bws, or pass "insight=Y" as a URL parameter to the Presto environment.

After capturing user data for use in Presto Insights, open Help>Insights Records and click the Download All button:

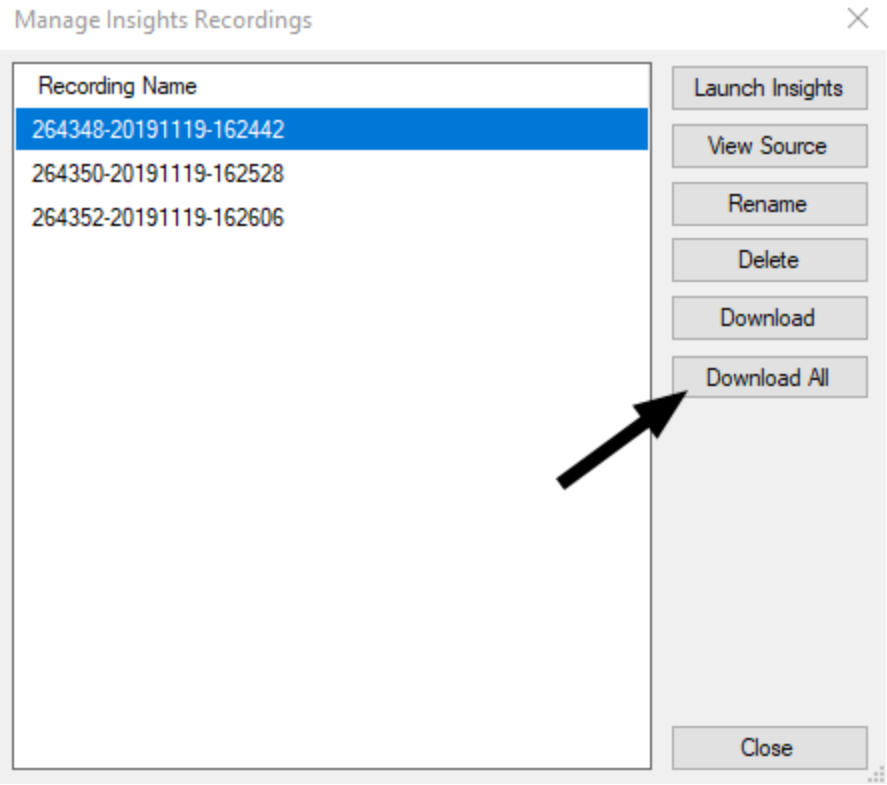


Figure 2-2. Option to Download All insight recordings.

After clicking Download All, you will be prompted to select where to save the insights recordings on your PC.

You can then use these recording files to create a new graph or import recording data to an existing graph.

Designer Access to Insight Recordings

You can access these easily through the Help menu (Help>Insight Recordings).

Running the Insights Tool

Once you've created a recording, you view them in Insights. You can start Insights one of two ways, you can either launch it through the Presto IDE, by accessing Help>Insight Recordings, and clicking "View Insight" on a recording. Or you can access it via the "Insights 1.0" start menu item created alongside the Presto item.

Presto creates a graph representation of how users have used the application, identifying which screens were accessed most frequently. The graph can visualize this data in different ways,

including a colored heat map, an opacity heat map, and a font heat map. There are configuration options to show the graph by visit or time spent.

Using the Tool

If the tool has been launched manually, you will need to create a new Insights Graph, this is done via File>New>Insights Graph.

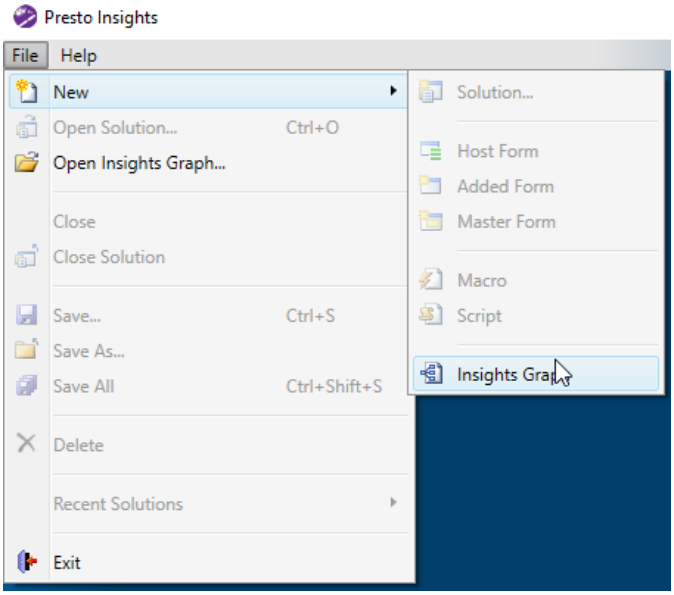


Figure 2-3. Creating a new Insights Graph.

Once you've created a graph, you can import recording in to it via the File>Import Recording menu item. It is possible to select multiple recordings to import at once using this menu.

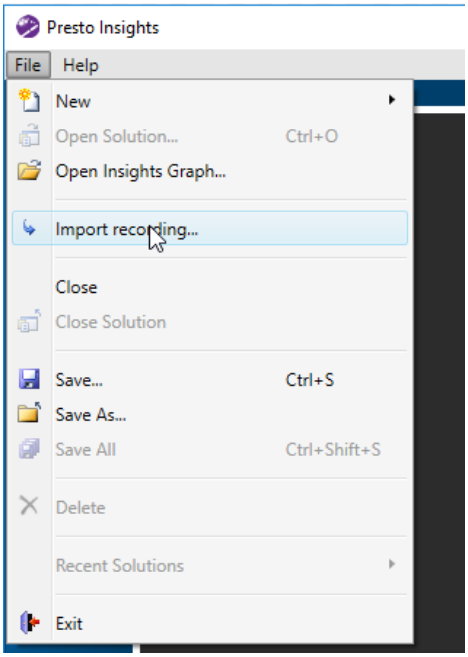


Figure 2-4. Importing a Recording.

Once you've selected your recordings to import, you need to specify a group to import them into (This allows you to display usage based on groupings such as role, user, or application area).

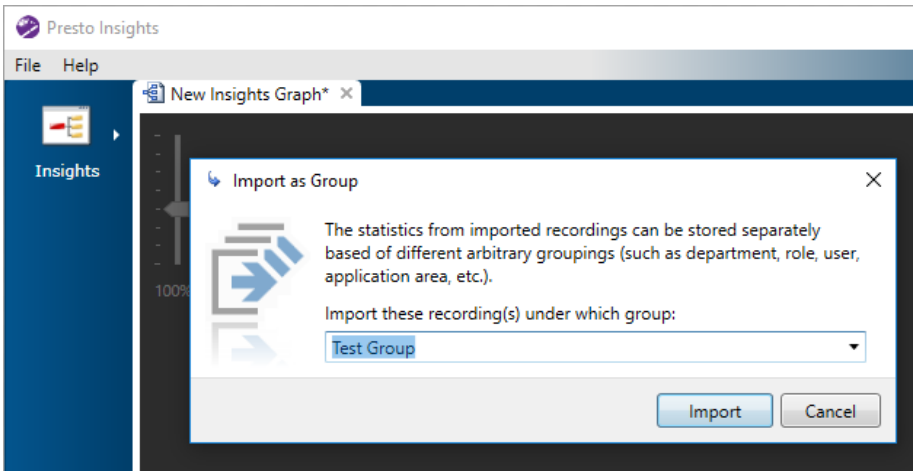


Figure 2-5. Specifying a group to import them into.

You will now have a graph of screens, as well as the connections between those screens (Where users came from and went to from each screen)

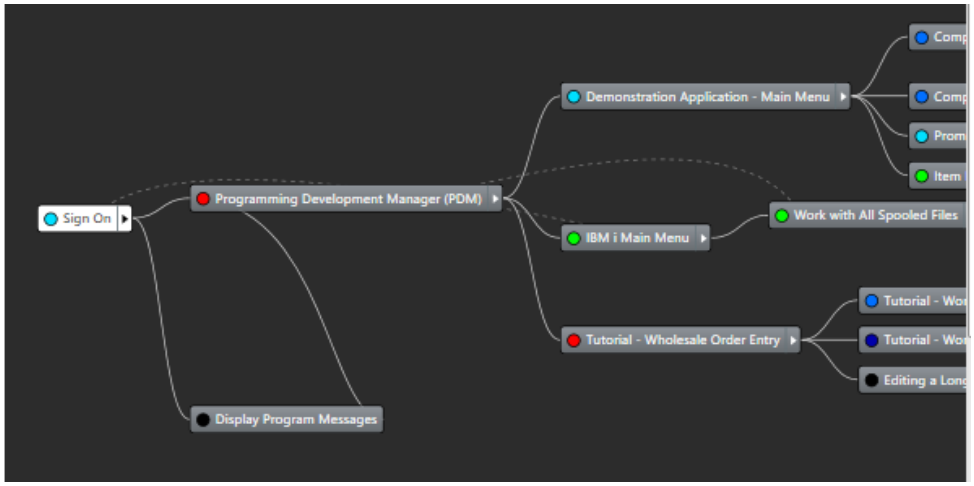


Figure 2-6. Insights Graph.

Available Visual Options

Color Heatmap

The heatmap is toggled by the red/yellow/green/blue icon:

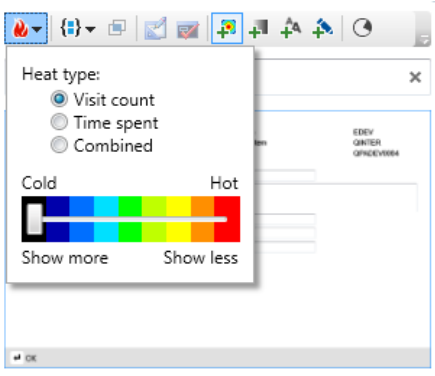


Figure 2-7. Heatmap options.

You can then utilize the flame dropdown to change specific options about the way the heatmap intensity is determined. The heatmap will allow you to get an overview of screen usage with a quick glance

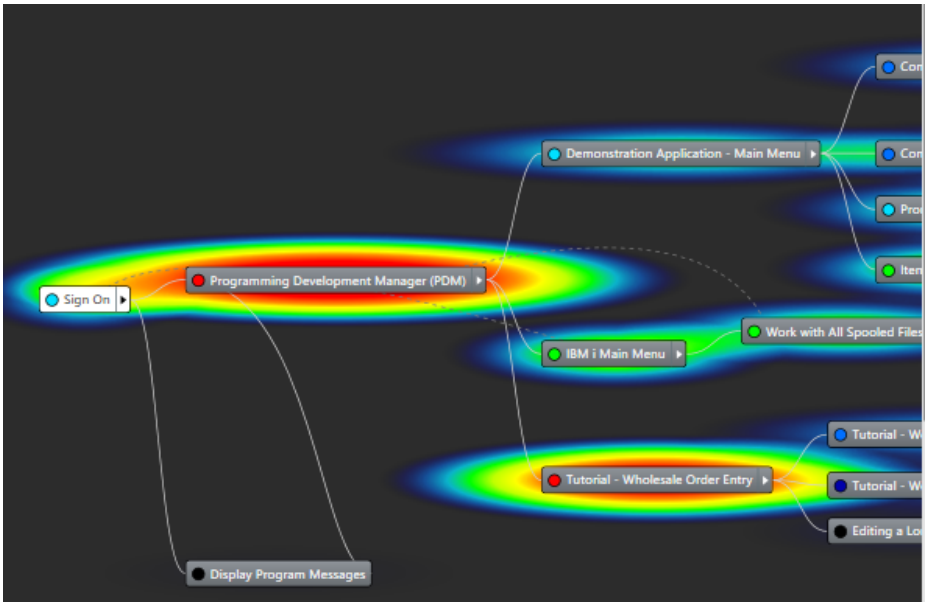


Figure 2-8. Heatmap Graph.

Opacity Heatmap

The opacity heatmap is similar to the color heatmap, except it uses transparency to represent the values as opposed to colors. It is toggled on using the greyscale gradient icon

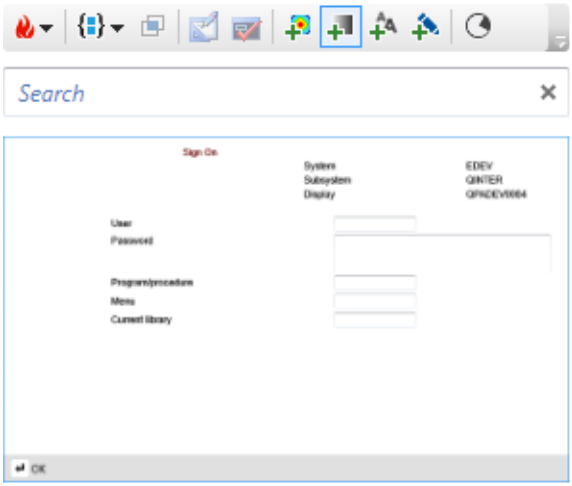


Figure 2-9. Opacity options.

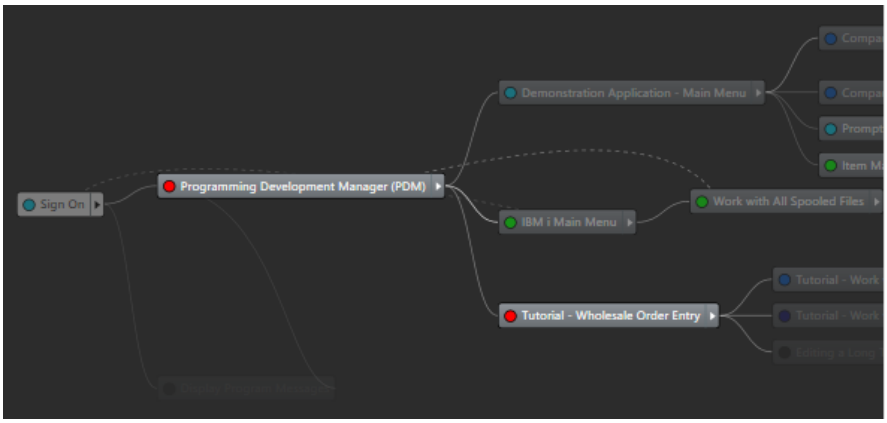


Figure 2-10. Opacity graph.

Font Heatmap

The font heatmap is similar to the color heatmap, except it uses the font size of the label to indicate values as opposed to colors. It is toggled on using the "AA" icon:

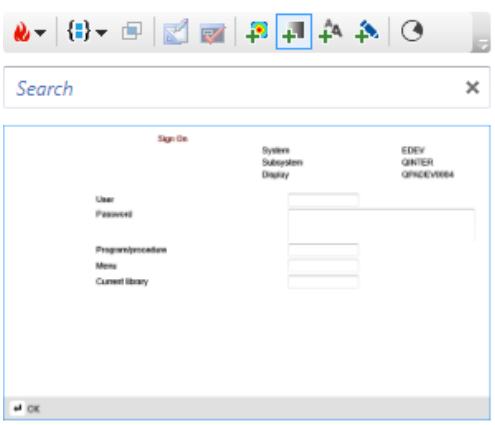


Figure 2-11. Font HeatMap options.

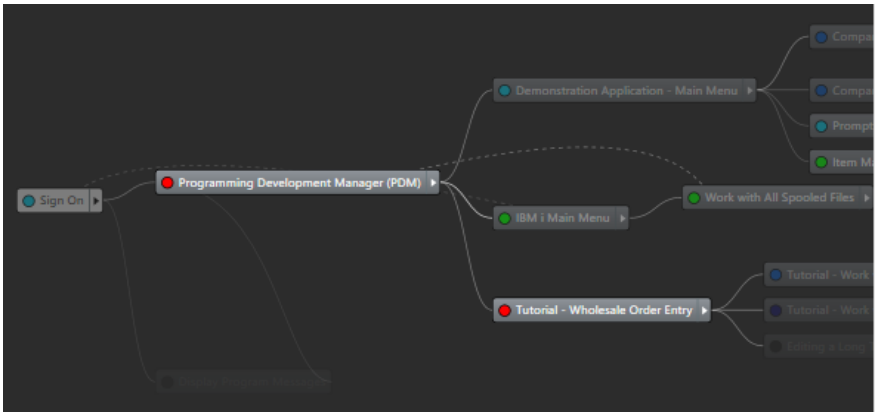


Figure 2-12. Font HeatMap options.

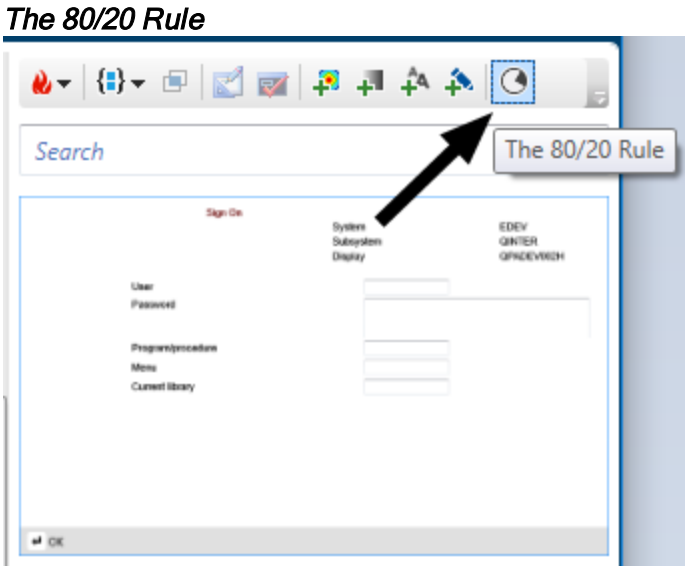


Figure 2-13. The 80/20 rule dialog.

This dialog allows you to identify those screens where the top 80% of your user traffic went. The general idea is that your users will be spending 80% of their time on approximately 20% of your screens, so these screens are a good candidate for modernization. The dialog will give you the top screens that 80% of your user traffic is to.

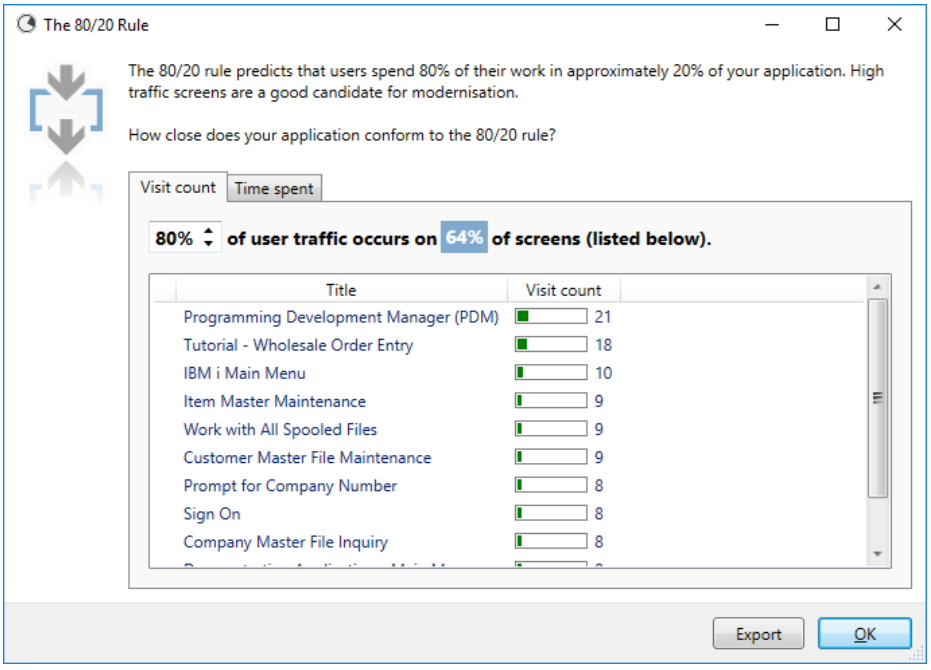


Figure 2-14. The 80/20 rule display.

There are two ways that you can report activity - screens visited and time spent.

Visit count. This tab will return the top 80% most visited screens, ordered by most visited. You can vary the user traffic percentage to return a different set of results.

Time spent. This tab will return the top 80% of screens where the most time was spent, ordered by most time spent. Note that if a user left a screen unattended and open during the recording, it may skew the accuracy of the time-spent statistics.

Note that clicking a screen in the 80/20 rule selects that screen node in the graph.

The data on this screen can also be exported to an Excel spreadsheet by clicking the "Export" button. This spreadsheet will contain the screen key (Top left text), screen title, total visits, and total time spent on that screen:

	A	B	C	D	E
1	Override?	Key	Title	Total Visit	Total Time
2		Programm	Programm	21	0:01:01
3		STEVEN	Tutorial - '	18	0:00:19
4		MAIN	IBM i Mair	10	0:00:08
5		ITMMNT	Item Mast	9	0:01:53
6		Work with	Work with	9	0:00:06
7		CSTMNT	Customer	9	0:00:15
8		CSTIQL	Prompt fo	8	0:00:13
9		Sign On	Sign On	8	0:00:20
10		DSPLAY	Company	8	0:00:09
11		DEMOMN	Demonstr	8	0:00:26
12		CMPINQ	Company	6	0:00:20
13		PT_CORDS	Tutorial - '	6	0:00:06
14		UPDATE	Company	6	0:00:23
15		PT_WWHS	Tutorial - '	4	0:00:03
16		Display Sp	Display Sp	3	0:00:03
17		EXAMPLE(	Editing a L	2	0:00:01
18		Display Pr	Display Pr	1	0:00:01
19					

Figure 2-15. Exporting Data.

Appendix G - License Key Server

License Server for Number of Users

Presto license keys can be purchased for a restricted number of users or unlimited users. Presto includes a license server subsystem (introduced in version 8.1) to track the number of users and enforce the number of users.

Here are some additional technical details about this change:

- The license server is installed to a library named XL_LCSRV.
- The license-server subsystem is called LICSRV. This subsystem uses port 11111.
- The license server is installed and started during the install process.

Presto Administrators can view the number of Presto seats currently in use by launching the Presto administrator interface. See "Presto Administrator Interface" on page 465 below for more details on this interface.

Only Presto Administrator sessions will see any mention of the license server. The Presto administrator sessions will get a notification if the license server is not active, if there's a communication issue with the license server, or if the seat count is exceeded.

Error Messages:

Error Connecting to License Server. The subsystem might not be running. Check that the LICSRVsubsystem is active.

ERROR: License Server denied Session Reset. This error can be the result of exceeding the seat count or an issue with the subsystem. Check the LICSRV subsystem is active and the administrator interface to verify the seat count is not exceeded.

ERROR: License Server denied Session Information. The seat count may have been exceeded. Please check the number of users in the administrator interface.

Error, running out of invalid program or library. This error is caused by the license server running from a library that isn't named XL_LCSRV or the program name has been changed from LICSRV.

Error, unable to retrive port to run on. The license server has not able to retrieve port information stored in the data area called PORT.

Error, unable to get lock on PORT data area. The license server has not able to retrieve port information because there's already a lock from another program data area called `PORT`.

Error, unable to get lock on install library. The license server cannot allocate the library and it's in use.

If you're starting `EXODUS55` in a start-up CL program, you may want to change the `STRSBS` command to also include the new license server subsystem: `STRSBS XL_LCSRV/LICSRV`.

License Server for Unlimited Seats

If you have an unlimited seat count then the license server is not required to access Presto. Turning off the license server will have no impact in your daily use of Presto. The only downside to turning off the license server is that the Presto Administrator Interface that allows you to verify the users who are connected will not be available. This interface was designed to help clients keep track of their seat utilisation. Having an unlimited seat count, the Presto Administrator Interface is most likely not something that you use.

Note: Turning off the license server will create various [Error] messages in the debug logs indicating that the connection to the license server is failing. These can be safely ignored.

Presto Administrator Interface

Presto includes an Administrator Interface for providing real-time monitoring of license usage. The Administrator Interface gives a quick overview of how many users are currently active within Presto.

To begin using the Admin Interface, open an Internet Browser session and enter this URL:

```
http://xxx.xxx.xxx.xxx:8022/presto/admin/
```

Replace the `xxx.xxx.xxx.xxx` with your own IBM i domain or IP address.

Here is an illustration of the interface:

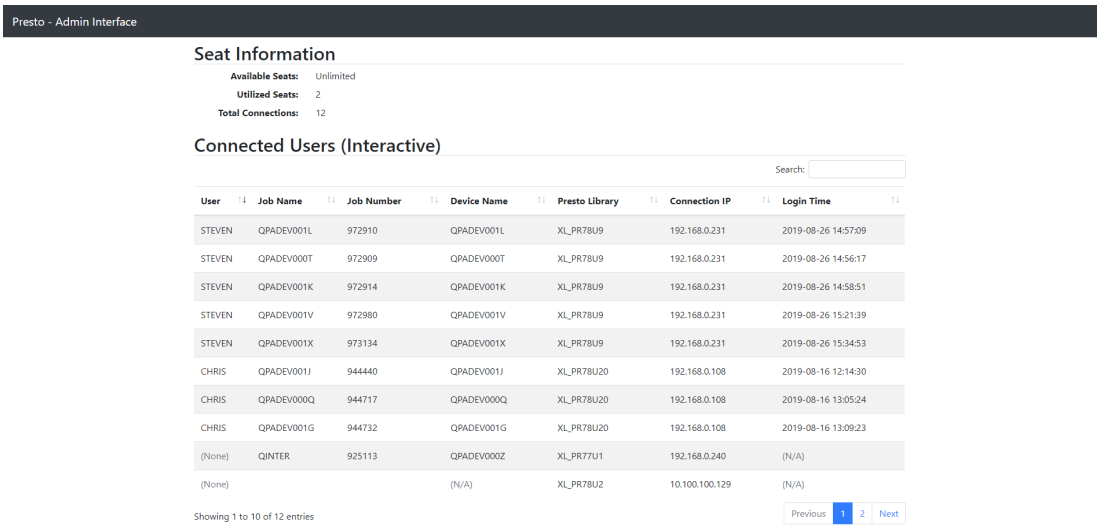


Figure 2-1. Launching the Presto Administrator Interface.

Seat Information

Here is an illustration of the Seat Information:

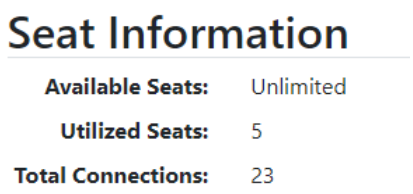


Figure 2-2. Summary of the Seat Information.

The Seat Information section lists the following three numbers:

- ***Available Seats***: The total amount of seats the currently installed license allows. This will either be "-1" if no license is installed, a number of seats if a license is installed with a limit, or "Unlimited" for unlimited licenses.
- ***Utilized Seats***: The amount of seats currently in use. This number is determined by counting how many unique users are logged in to Presto at the time of the request.
- ***Total Connections***: The amount of total Presto connections active to the license server. This includes sessions that are currently at the sign on screen, which will not show up under the below user list.

Connected Users (Interactive)

The "Connected Users (Interactive)" section has a table of all the currently signed on users, spanning all environments on the system that are communicating with the license server. The table consists of multiple columns of data about the connection:

- ***User***: The name of the user currently signed in.
- ***Job Name***: The job name of the user's interactive job.
- ***Job Number***: The job number of the user's interactive job.
- ***Device Name***: The device name in use by this session.
- ***Presto library***: The library that the job is currently running out of.
- ***Connection IP***: The IP address that the user is connecting from.
- ***Login Time***: The time that the user logged in to Presto.

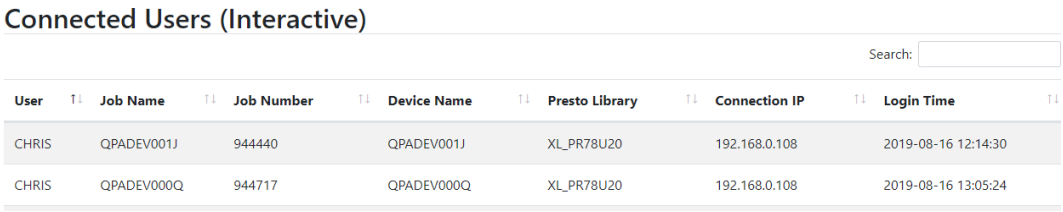


Figure 2-3. Connected Users.



Older versions of Presto do not send extended session information, so the Library, IP, and Login Time columns will be blank for those sessions.

Error Messages

When running into an error, the administrator panel will show a modal dialog stating that "There was an error fetching the status information." and a message in smaller text at the bottom. This message will be the technical reasoning behind the failure.

Below is a list of the know error messages and their cause:

Error Message	Reason	Resolution
Server denied SetProduct	The license server that is being run does not support the status retrieval protocol.	Please restart the license server to load the latest version that was installed with this version of Presto.

Troubleshooting Questions

1. ***Why is the user data missing, such as the Presto library and connection IP?*** If the user is running an older version of Presto alongside a newer version, entries in the table may be missing the last three columns. These entries from earlier versions of Presto will only show the username, job number, and job name..
2. ***Why are there QINTER jobs shown without a user name?*** If a user is on the sign on screen without yet signing onto Presto, the entry in the user table may consist of only the username. This is expected behavior, as there is no interactive job associated with this session. Once the user has successfully signed in, the license server will display all the information and show the username.
3. ***How many License seats does a Presto User Occupy when signed into Multiple Browser Sessions?*** License seats are based on the number of users. They are not based on the number of sessions or browser tabs that a user has open. For example, if 4 users eachhave 5 sessions open, only 4 seats are taken up.

Seat Count Utility

Presto ships with a utility called `SEATCOUNT`, which allows administrators to get an idea of how many active Presto seats are currently in use.

Calling the Utility

The utility is called from a green-screen session:

```
CALL XL_PRESTO/SEATCOUNT
```

(assuming that your Presto library is called XL_PRESTO)

This information is output to the screen:

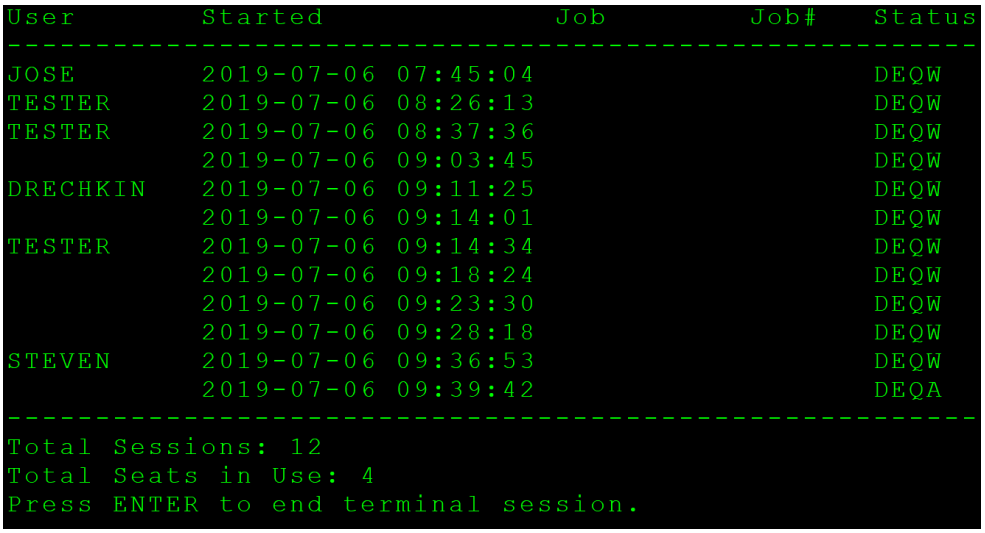


Figure 2-4. Output of Seat-Count Utility (job/library redacted)

Advanced Usage

The utility can also store information in data areas for more advanced customers. You will need to manually create the SEATMAX and/or SEATCUR data areas in your Presto library. The data area named SEATMAX needs to be created in the Presto library you want to track. The data areas will be updated any time you run CALL SEATCOUNT, allowing you to automatically record user information and integrate with user-created utilities for monitoring.

Here is a version of the command that you can use:

```
CRTDTAARA DTAARA(XL_PR100/SEATMAX) TYPE(*CHAR) LEN(32)
```

where XL_PR100 is the environment you want to track and *CHAR is the data type.

If you want to keep the tracking information in the data area SEATMAX up-to-date, you can create an automated job to run CALL SEATCOUNT.

SEATMAX

If the SEATMAX data area exists, its value will be compared against the current in-use seats, and updated if there are more seats currently in use.

SEATCUR

This data area will be updated with the current in-use seat count on each run of the utility.

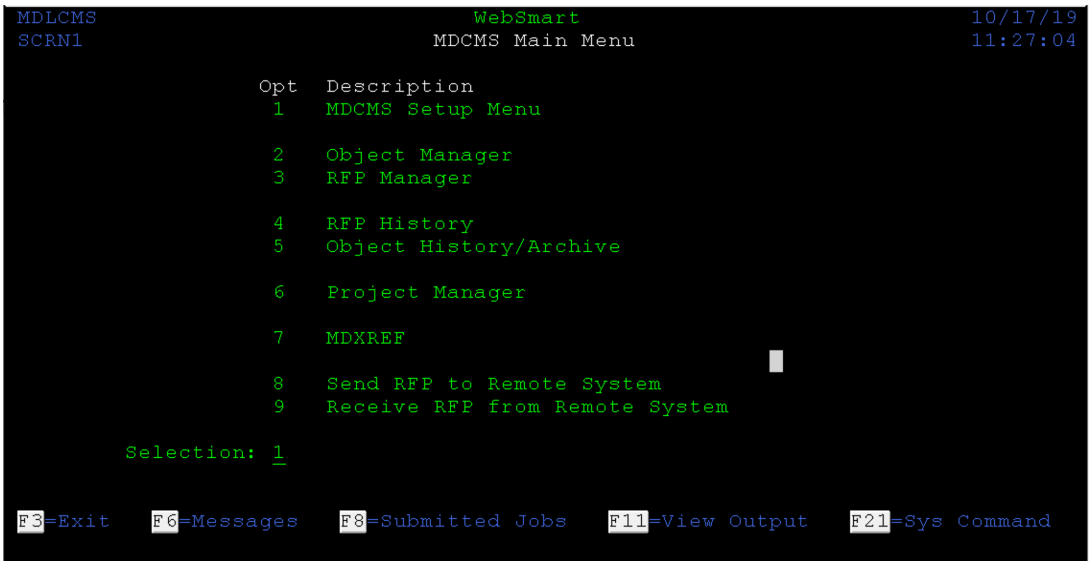
Appendix H - MDCMS Change Management Integration

Midrange Dynamics Change Management (MDCMS) provides tools for promoting changes from a development environment to production. MCDMS integration has been implemented in this version, allowing Presto to register change management export files in MDCMS.

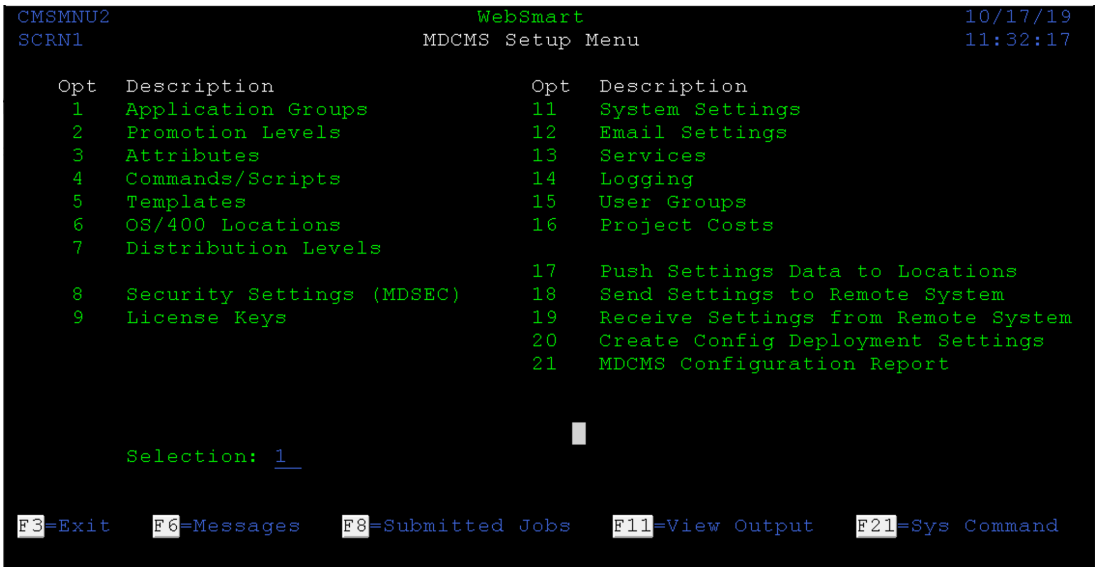
How to Configure MDCMS to work with Presto Change Management

To integrate Presto with MDCMS, the **Application, Level, Attribute, and Project** need to be configured in MDCMS.

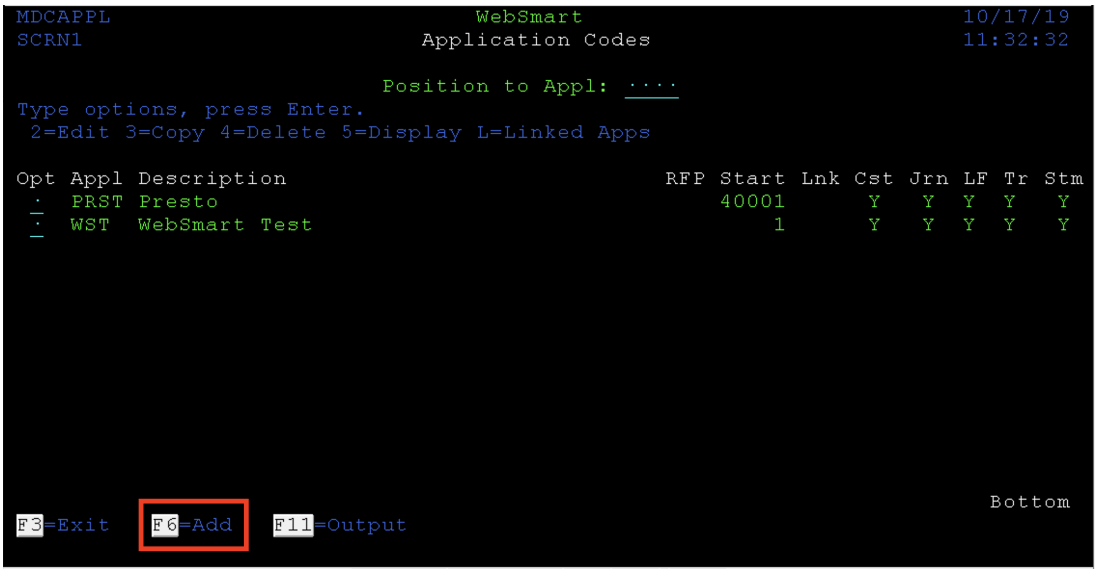
As a first step, type **MDCMS** and hit enter.



Then type Selection: **1** to open **MDCMS Setup Menu**.



Then you will have to configure **Application Groups**, for that type **1** into Selection.



To Add a new Application you will have to hit **F6**.

```
MDCAPPL                          WebSmart                          10/17/19
SCRN2                        Application Code                      11:36:07


Application Code:     PRST  Presto

RFP Start Index:    40001  RFP Last Created:

Automatically Reapply:
 Constraints  . . .  Y  Y/N
 Journaling . . . .  Y  Y/N
 LF Members . . . .  Y  Y/N
 Triggers . . . . .  Y  Y/N


Update Object Description With:
 Attr / User Defined Attribute. . Y  Y/N
 RFP # 1 / Object Control Level . C  N=None, F=From, O=Origin, C=Current
 RFP # 2 / PTF  . . . . . . . . . O  N=None, F=From, O=Origin, C=Current
 Appl, Level  / APAR  . . . . . . Y  Y/N
 Proj, Task, Subtask / LICPGM . . Y  Y/N

Enter=Confirm    F12=Previous
```

There you will specify and note these values for future reference -**Application Code** and **Name**.

Then you can hit **F3** to return to **MDCMS Setup Menu**.

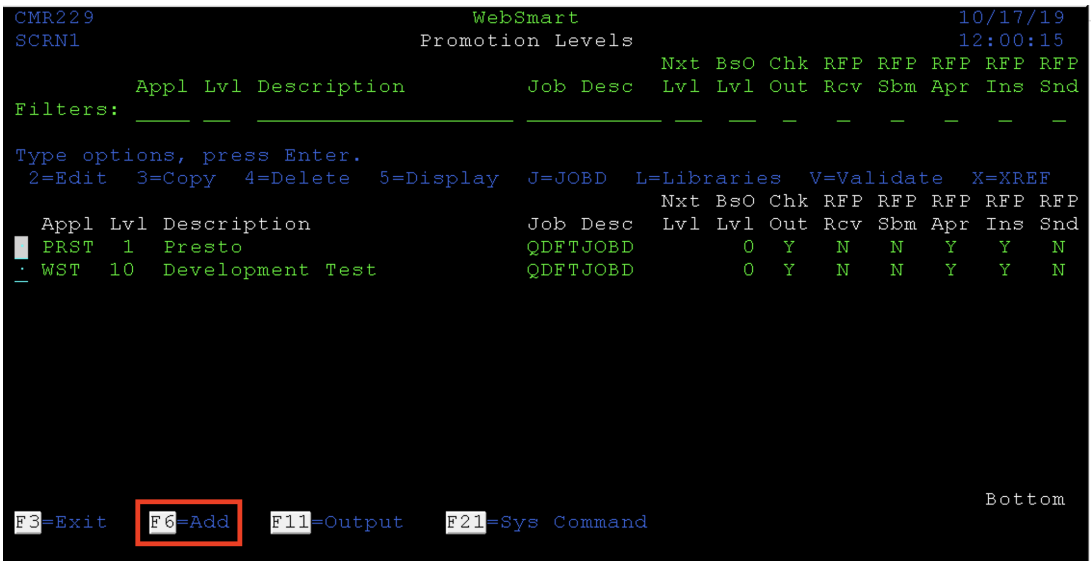
```
CMSMNU2                          WebSmart                          10/17/19
SCRN1                        MDCMS Setup Menu                      11:32:17

   Opt  Description                 Opt  Description
    1   Application Groups          11   System Settings
    2   Promotion Levels            12   Email Settings
    3   Attributes                  13   Services
    4   Commands/Scripts            14   Logging
    5   Templates                   15   User Groups
    6   OS/400 Locations            16   Project Costs
    7   Distribution Levels
                                    17   Push Settings Data to Locations
    8   Security Settings (MDSEC)   18   Send Settings to Remote System
    9   License Keys                19   Receive Settings from Remote System
                                    20   Create Config Deployment Settings
                                    21   MDCMS Configuration Report


        Selection: 1

F3=Exit    F6=Messages    F8=Submitted Jobs    F11=View Output    F21=Sys Command
```

After that, you will have to configure **Promotion Levels.** Type **2** into Selection.

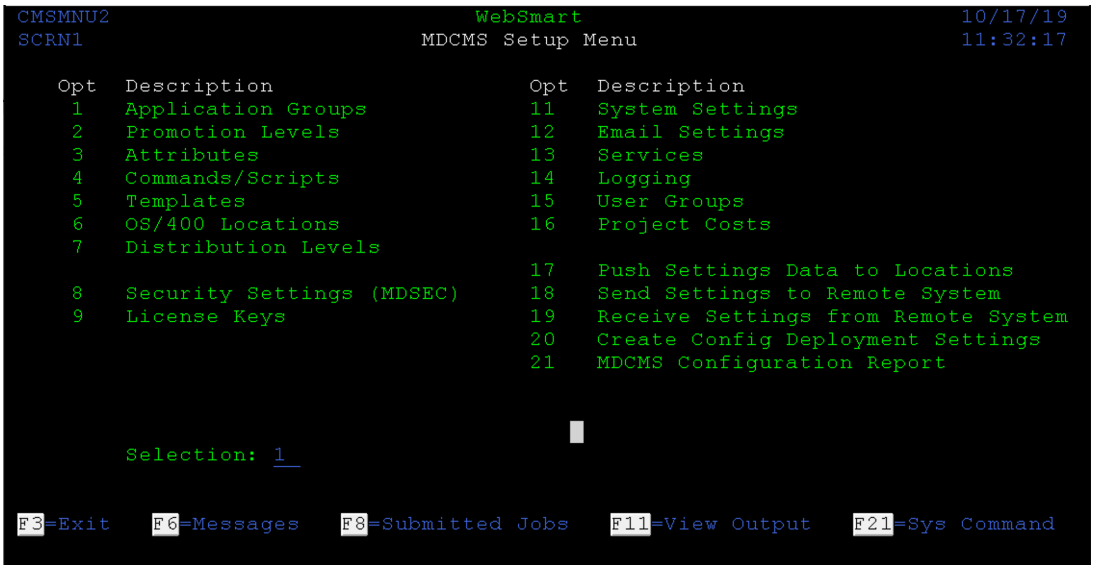


From there, hit **F6** to add a new **Application Level**.

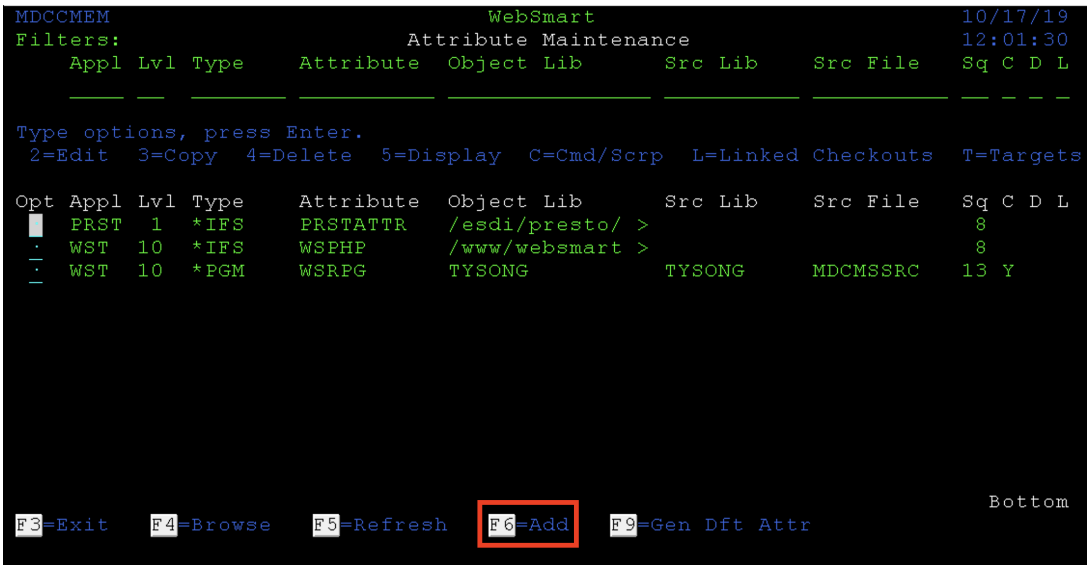


To specify a previous configured **Application Code**, select numeric **Install Level** as well as other options.

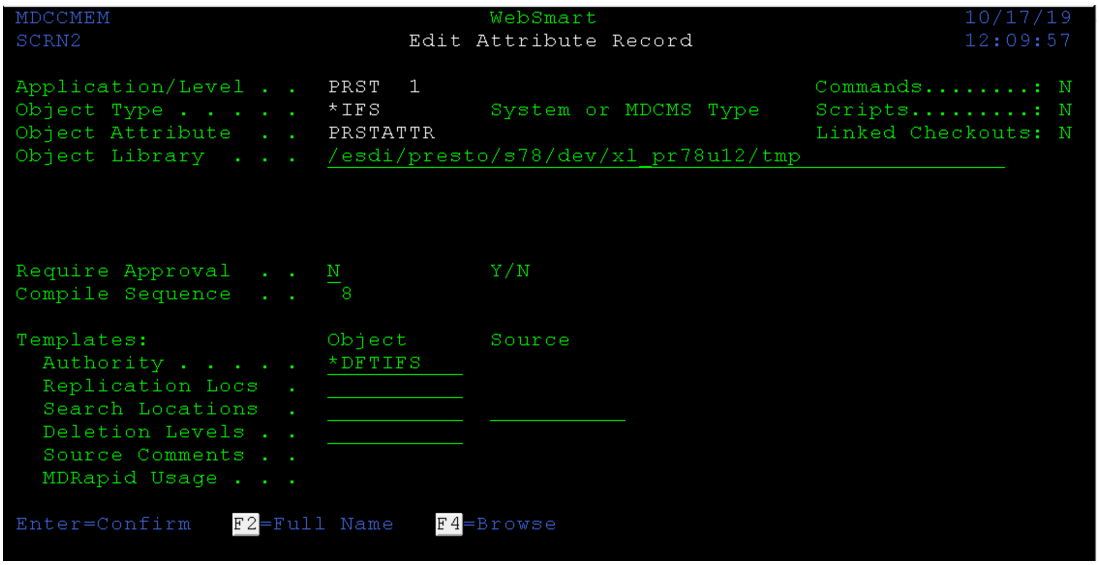
Then you can hit **F3** to return to **MDCMS Setup Menu**.



Then you will have to configure **Attributes**, for that type **3** into Selection.



From there, hit **F6** to add new **Attribute**.

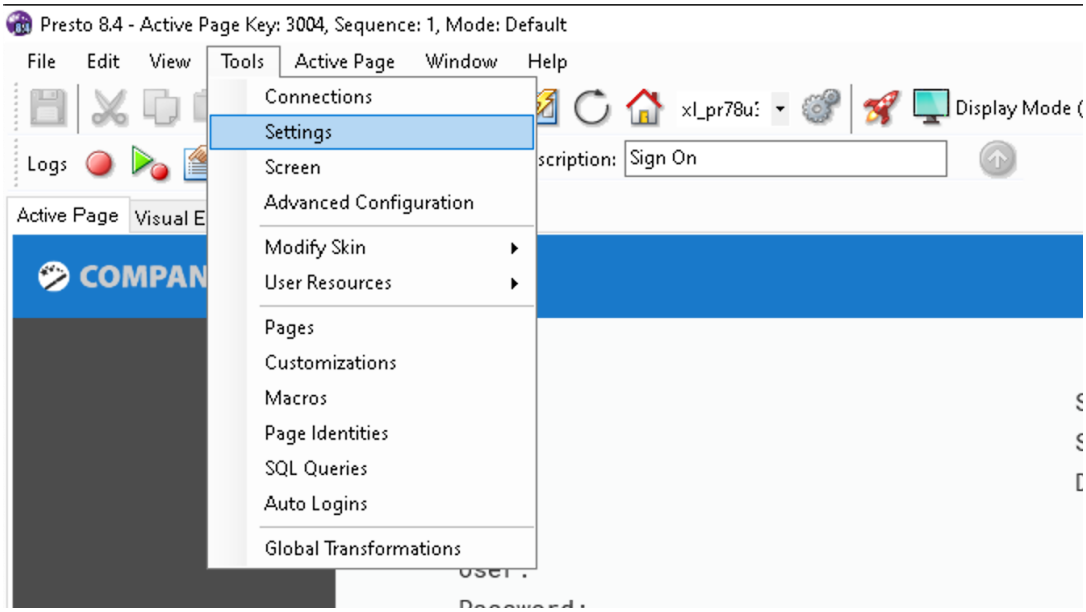


Specify previous configured **Application Code**, set numeric **Level**, set object type to ***IFS**, set previously configured **Attribute** and **Object Library** of where xml files will be stored.

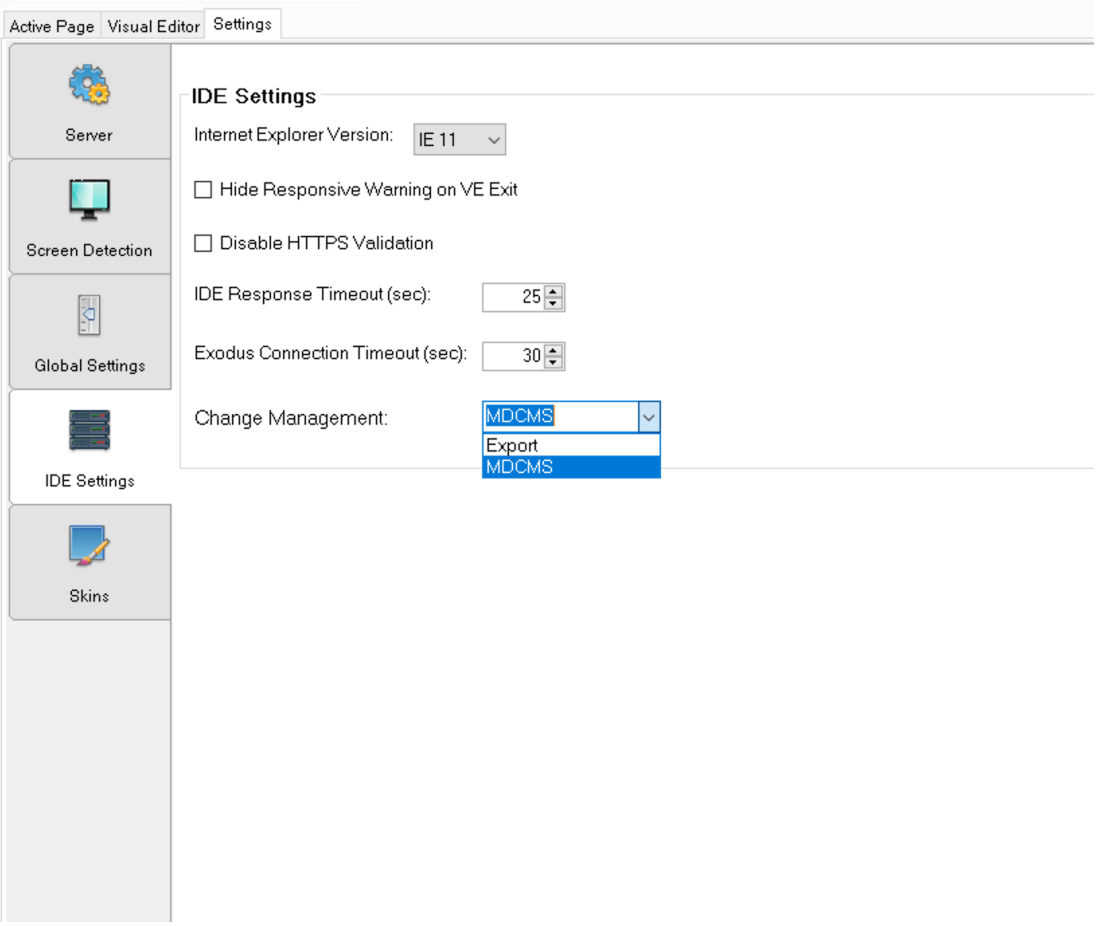
After completing these steps, you can start using MDCMS in Presto.

How to use MDCMS in Presto

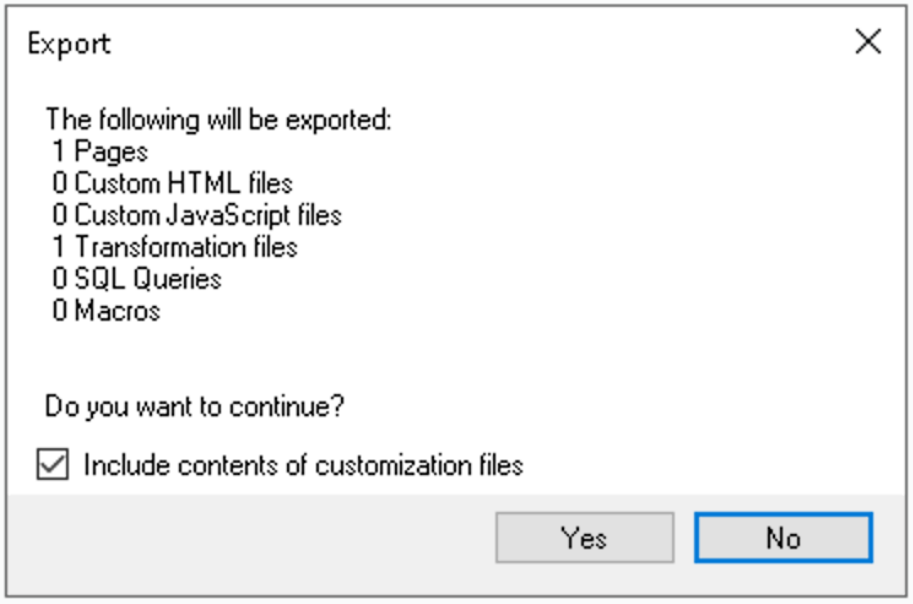
Open Presto IDE and Go to **Tools>Settings**



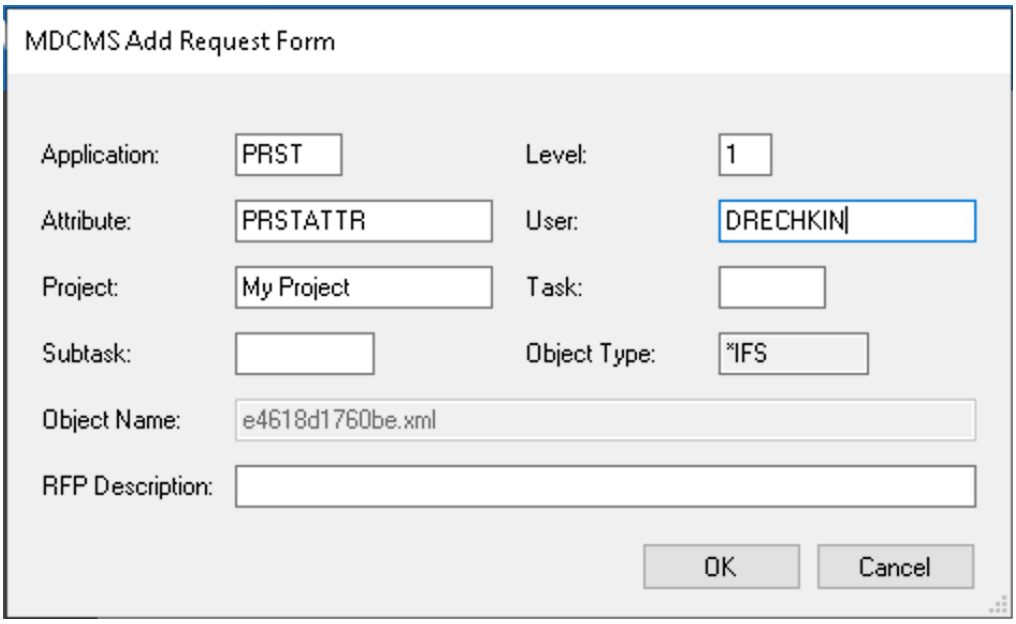
Then choose **IDE Settings** tab and switch **Change Management** to **MDCMS**.



Then you can export screens in a regular fashion via **Manage Pages** or **Panel Options** and the first Confirmation Dialog will remain identical:



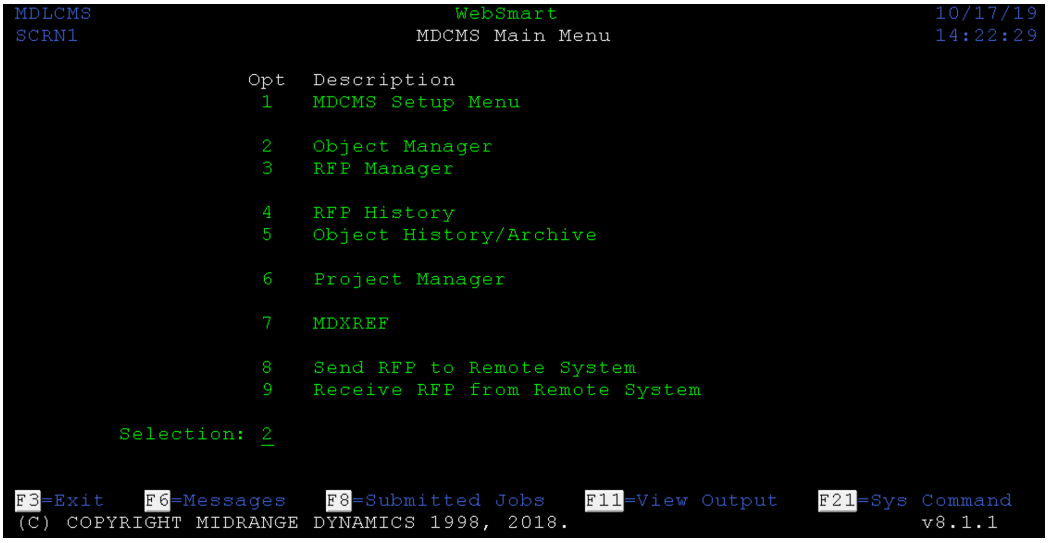
After hitting **Yes**, you will be presented with **MDCMS Add Request form**



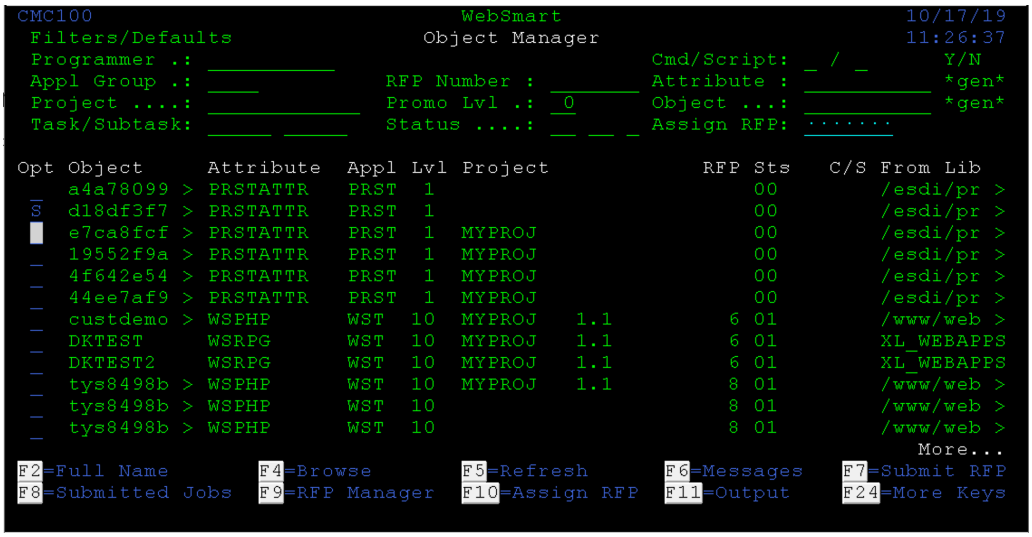
Application, Level, Attribute are the same values we've configured in MDCMS above.

These values will be stored and pre-filled in advance for the next time you use it.

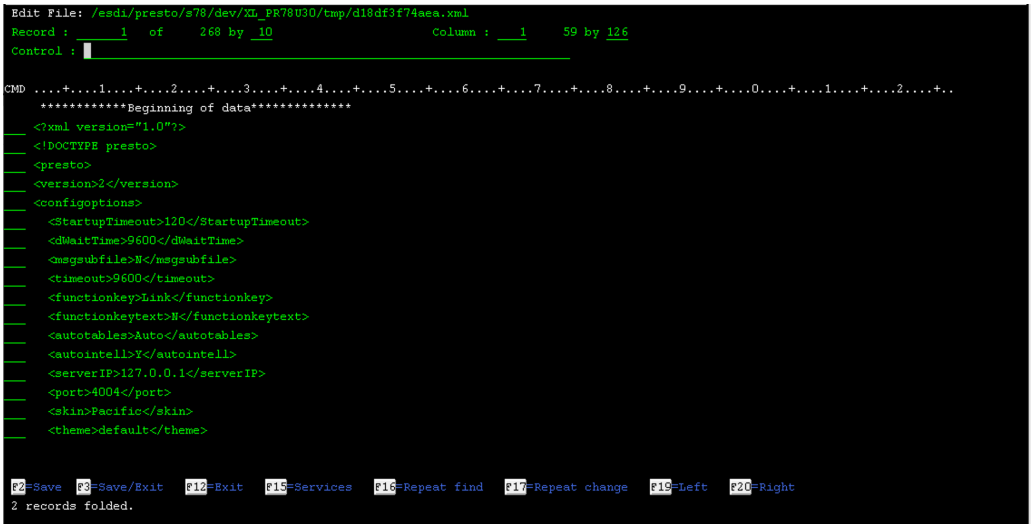
Once one or more screens have been successfully exported in Presto, information about those screens can be found in MDCMS.



Type **2** into Selection field to open **Object Manager**.



Then you can type **S** next to recently exported object.



You will be presented with the source code of the exported file.

Appendix I - Health Check Tool

The Health Check tool can provide valuable information about the health of your Presto environment, allowing you to identify and address issues before they become more serious problems.

The Health Check tool is designed to report on the following health issues in the environment:

- **Damaged objects in the library.** This could include any object that has become corrupted or is no longer accessible due to issues with the file system or storage. It is based on damage as reported by the OS.
- **Orphaned messages in the data queue.** This could include messages that have been in the queue for longer than the `dWaitTime` value in config.bws, indicating that they are no longer relevant or have not been processed properly.
- **Data queue fill level.** It reports the total number of messages on each data queue, along with the maximum number of messages that data queue can hold, and an approximate percentage value of how full the queue is. This helps to identify if the data queue is nearing capacity and to take appropriate action to prevent performance issues.
- **License Server status.** It checks whether the License Server is available, and whether licenses are invalid or expired. For licenses with unlimited seats, the Health Check tool will not respond with an error if the license server is inaccessible.
- **Active Subsystem status.** It checks for an active subsystem to launch PR_HUB jobs in. By default, PR_HUB jobs are launched in QUSRWRK, but the environment can be configured to use a custom subsystem (see https://www.myfreschesolutions.com/kb?kbnum=15438 for details). If a custom subsystem is configured, the Health Check tool will check whether that subsystem is active.

Object Status Reported

The health-check tool uses the following object statuses to report the state of objects in the system:

Object Status	Description
unreadable	The tool could not access the object, possible reason could be authority issues. An `errorCode` property is included with the object containing the IBM i exception

Object Status	Description
	code for troubleshooting.
healthy	The object is accessible and there are no issues. If a data queue, then there are no orphaned messages and the queue is below 90% full.
orphans	Only used for data queue objects. Indicates that one or more orphaned messages were found on the data queue.
dqfull	Only used for data queue objects, Indicates that the data queue is more than 90% full. In this case, the queue could be too small and it needs to be recreated with a larger maximum size or multiple data queues are needed to handle the load.
damaged	The object has been reported as damaged, and it needs to be deleted and replaced.

Non-Object Statuses

The health-check tool reports on the status of the connection to the license server, license key, and active subsystem. The statuses are as follows:

Status	Description
healthy	No issues were reported by the tool.
failed	The attempt to connect to the license server or active subsystem for PR_HUB jobs failed.
invalid	There are no valid license keys with the product code for Presto.
expired	The temporarily license key has expired.

Running from the Command Line

The health-check tool can be accessed from the command line using the following call:

```
CALL XL_PRESTO/HEALTHCHK
```

This command can also be called via QSH:

```
QSH
/qsys.lib/xl_presto.lib/healthchk.pgm
```

The summary is output to the terminal:

```
Command Line Health Report
Library: XL_PR91U10
Objects of Concern:
        maxMessages: 3280
        name: PR_CONVQ
        numMessages: 6
        numOrphanedMessages: 6
        percentUsed: 0
        status: orphans
        type: *DTAQ

1 issues found in library XL_PR91U10
```

It is important to note that in the examples "XL_PRESTO" should be replaced with the name of your library.

Running the Tool via HTTP Request

Running the health-check tool via HTTP requests is a convenient and efficient way to monitor the health and performance of your Presto environment.

A healthcheck route is available through the Presto httpd.conf file that allows for requests to run the tool and receive a JSON report in return. This report can be accessed by navigating to a specific URL in a browser or via an AJAX request.

To run the tool from the browser, you can simply navigate to the URL of your Presto environment, but with the end of the route being healthcheck instead of Presto. For example:

```
http://www.yourserverip.com:8023/presto/healthcheck
```

You will see the JSON report in the browser but as plain text.

The HTTP response code returned by the health-check tool also provides useful information about the status of the environment. A successful response code (200) indicates that the environment is healthy, while a non-successful response code can indicate that there are issues present:

Object Status	Description
200	The environment is healthy based on reported metrics.
500	One or more issues are found in the environment.
503	An error prevents the health-check tool from running. The JSON report should still return with an Error property containing a description of the problem.

One of the advantages of running the health-check tool via HTTP requests is that it allows for automated monitoring. By making an AJAX request to the healthcheck route, you can programmatically check the status of your Presto environment on a regular basis and take appropriate action if any issues are found.

The same URL can be used to make AJAX requests to the tool. The response data has a mime-type `text/json`, and the body should always be a valid JSON object. Here is an example:

```
jQuery.ajax({ url: 'http://www.yourserverip.com:8023/presto/healthcheck', complete:
function(data) { console.log(data.responseJSON.Report) }})
```

This script sends an HTTP request to the specified URL and logs the response to the console when it's loaded.

The `success` property indicates whether the tool ran successfully or not. If the value of the success property is "true", that means the tool ran successfully and there will be a report property with additional information about the objects in the library. If the value of the `success` property is "false", then the tool did not run successfully and the `messageId` and `messageText` properties will provide more information about the failure, such as an HTTP response code and a human-readable explanation as to why the tool failed.

The report includes an array called `allObjects`, which lists all the objects in the library with their status information. There is also an array called `objectsOfConcern`, which lists any objects that have a status other than "healthy". If there are no objects with issues, the `issueCount` property will be 0 and the `objectsOfConcern` array will be empty. Objects of type "*DTAQ" have additional fields such as "maxMessages", "numMessages", "numOrphanedMessages", and "percentUsed" that describes the detailed state of the queue.

Here is a shortened example of the report:


```
{
   "success" : true,
   "report" : {
      "libraryHealth" : {
         "allObjects" : [
            {
               "name" : "CRTDTAQS",
               "status" : "healthy",
               "type" : "*PGM"
            },
            {
               "name" : "HEALTHCHK",
               "status" : "healthy",
               "type" : "*PGM"
            },
            {
               "maxMessages" : 3280,
               "name" : "PR_CONVQ",
               "numMessages" : 244,
               "numOrphanedMessages" : 238,
               "percentUsed" : 7,
               "status" : "orphans",
               "type" : "*DTAQ"
            }
         ],
         "issueCount" : 1,
         "libraryName" : "XL_PR90",
         "objectsOfConcern" : [
            {
               "maxMessages" : 3280,
               "name" : "PR_CONVQ",
               "numMessages" : 244,
               "numOrphanedMessages" : 238,
               "percentUsed" : 7,
               "status" : "orphans",
               "type" : "*DTAQ"
            }
         ],
         "objectCount" : 301
      }
   }
}
```

Appendix J - Frequently Asked Questions

This appendix is a compendium of frequently asked questions asked during Presto webinars, or sent to the sales and support teams.

Question and Answers

Q: Is there a way to 'slowly ' roll out the converted screens, say by dept.?

A: Yes. You can set up a development environment and then a production environment, letting you modernize at your pace. You can also enhance different sections of your screens over time.

* - *

Q: WE are <SLOWLY> going to a SOA environment. Mainly, it is aimed at replacing the IBM i with .NET on clustered servers, but we want to show that the IBM i can also play in that area. Any hints on doing that with old school RPG programs? (We still have some).

A: You can use AJAX to call RPG programs to retrieve portions of your data to display in your pages.

* - *

Q: Care to recommend some sort of HTML-for-Dummies book? Need all the help I can get.

A: Try www.w3schools.com . This site is a great place to learn the basics for a lot of web technologies, including HTML, CSS, JavaScript, AJAX and XML.

* - *

Q: Hex for ASCII or hex for EBCDIC?

A: ASCII. But don't worry about it! We hide all that complexity.

* - *

Q: What is the learning curve for an old IBM i programmer?

A: Presto's learning curve is very quick. It would take you days, not weeks, to become proficient at developing/enhancing screens using the Presto Designer. HTML is very quick to learn as well. Though it may take a while to truly master HTML, you can learn the basics in just a couple of hours.

* - *

Q: Presto is not a green scraper?

A: Presto uses the 5250 data stream to convert the data to HTML.

* - *

Q: In Presto, are standard images included in the software?

A: Presto is shipped with skins. Skins are the look and feel for your pages and each skin ships with a certain amount of images including header images and images for things such as buttons, links and calendars.

* - *

Q: Does Presto work with Cobol?

A: Presto works with RPG or Cobol. Presto does not need the DDS or source code.

* - *

Q: What about system screens like Work with Output Queues or Display Message?

A: With Presto, your whole system is Web enabled. You can run WRKACTJOB or any other program, menu or system screen. You can also enhance those screens if you wish.

* - *

Q: Does your RPG code need to be updated to RPG IV? We have older code.

A: There is no need to update any of your existing code to Web enable your RPG programs using Presto. It does not touch any existing code - it rides on top of your existing applications.

* - *

Q: Are there any problems interfacing with JDEdwards?

A: Shouldn't be any problem, since the result is a 5250 data stream.

* - *

Q: Does this date widget work for non-date fields and/or old 6.0 formats?

A: The date widget can format the date into any format that your program expects. Presto does not care if your fields are proper date fields or simply numeric values

* - *

Q: With menus, can you use icons instead of links to click?

A: Absolutely.

* - *

Q: Do you have any suggestions for getting date, times and number shift validation in text fields in HTML via JavaScript?

A: Yes. The best approach is to use jQuery, a JavaScript library that provides lots of routines to make this kind of validation easy to add. Also, within Presto we provide JavaScript functions for number shift validation.

* - *

Q: Is the system pricing (license account) for the developer license only? Does the end user needs to have a license as well?

A: Presto is available by tier that includes a specific number of concurrent seats as well as an unlimited seat version.

* - *

Q: Where is the web server installed?

A: The Apache web server that Presto uses to serve the pages is shipped with the IBM i. It runs on your IBM i. Note that this is standard software that comes with the operating system. You do not have to buy any licenses for additional products from IBM.

* - *

Q: Can Presto be used in S/36 environment?

A: Presto should run correctly in the S/36 environment, since it simply uses the 5250 data stream.

* - *

Q: Can I see the underlying XML?

A: Presto generates HTML, not XML, and you have full access to it.

* - *

Q: Can Presto be packaged with a vendor application and made to work only with that application? Or is it a system-wide tool to replace Telnet and work with all screens/applications?

A: Yes, this is possible. Although Presto is a system-wide tool that web enables your entire system, you can make some simple changes to only make your application available. You would just need to create custom HTML in Presto for whatever the default pages are when people log to their IBM i and have it go directly to your application.

You will also need to make sure you remove any command line access if it is on any of your screens, which you can do by customizing pages in Presto. In Presto, you can also add exit points and redirect people to any URL, so if the traditional sequence of exiting your application is to go to a command line you can change that via Presto.

* - *

Q: Does Presto support 132 character screens and switching between 132 and 80 screens?

A: Presto supports 132 and 80 character screen. If your applications switch between the two, Presto will handle that as well.

* - *

Q: Do all screens have to reviewed/updated in the Presto Designer or will they inherit a default look?

A: Presto will assign a skin, or 'look and feel' to each screen by default. Out of the box every screen on your system will inherit the same look. You will want to review each screen to review how the default settings are handling that panel, and to decide if there are enhancements that you want to make to that screen.

* - *

Q: Where and how does the application get launched?

A: Applications are launched using a Browser and a specific URL. The program objects are running on the IBM i as they would if you were calling them from a 5250 session.

* - *

Q: Can this work with hand held scanners?

A: We have not yet tested it, however, if they run mini-browsers then I don't see any reason why it does not. Presto can handle any 'client' that uses the HTTP protocol.

* - *

Q: Can buttons or hyperlinks be added to Presto screens that call WebSmart programs?

A: You can call WebSmart programs from Presto programs. You are not limited to just a call, you can pass parameters to WebSmart programs using information from the screen. You can also use WebSmart programs in conjunction with AJAX to seamlessly integrate new functionality in your screens.

* - *

Q: In addition to converting 5250 to HTML, is Presto also an IDE? Would we use Presto to design a new screen from scratch?

A: Presto is designed to work with programs already built. Our software development tools such as WebSmart, WebSmart PHP, Clover and ProGen will provide the ability to develop new programs.

* - *

Q: Is Presto licensed ($$$) based per user or site?

A: Presto has a couple of options, license based on tier and each tier includes a specific number of user seats as well as an unlimited user license based on tier.

* - *

Q: Our DC application uses handheld that use the standard 5250 screen except the screen display is sized for the upper left corner of the screen only. It works for 5250, I just don't know how the browser will work.

A: You can configure the CSS or 'style sheet' to run on handheld units provided the hand held units have the internet and a browser available to them.

* - *

Q: Can drop downs be used for subfile options?

A: You can use drop downs for sub-file options, as well as icons for the end user to click on.

* - *

Q: How hard is it to convert a "drop subfile" to one line across the page? (i.e. "drop" was necessary due to 80 character limitations.)

A: All of your screens are instantly converted, including drop subfile screens.

* - *

Q: With menus, can icons be clicked on instead links?

A: You can use icons as links in menus or any other image.

* - *

Q: Are program changes manageable by source/object control (SOX requirement)?

A: Program changes are stored as text files on the IFS, so you could place these under change management.

* - *

Q: Is there a performance hit running an application via Presto vs. green screen?

A: Programs running in Presto have some conversions taking place. There is very little if any performance difference between running your programs on a 5250 screen and through Presto. In fact, since Presto apps use no interactive processor you should see better performance in many cases.

* - *

Q: When you need to add a field/make a change, what do you modify? The original RPG/DDS source? Presto? Both?

A: You make the change in Presto usually, unless you need to add a field to your green screen application anyway. By using AJAX technology you can bring in other fields from the file even if they were not used in the original application.

* - *

Q: Auto Login works with enterprise level authentication?

A: It assumes you've already logged in to a web app securely- so, yes.

* - *

Q: As an ISV if I create a set of Presto "updated" programs can that be easily deployed to our customers (assuming proper Fresche licensing for the customer)?

A: Yes. You can send them one library.

* - *

Q: Is the additional language cost extra?

A: Presto supports multi-languages and multi-environments.

* - *

Q: Currently use MKS - Implementer for Change Management. Our screens/programs still change on a regular basis (weekly). Would you change DDS and then reformat in Presto?

A: Yes

* - *

Q: What about system screens like Work with Output Queues or Display Message?

A: It handles those too.

* - *

Q: Does Presto have the ability to pull data from multiple screens in the background and present it on a single screen?

A: Not just yet, however this is a feature that we will add. You can take a single screen and separate it into tabs.

* - *

Q: What is the lowest level of OS/400 that is needed for Presto?

A: Currently Presto needs to have a minimum OS of V6R1.

* - *

Q: Once converted by Presto, can we delete our DDS?

A: Presto does not require DDS, however, you will want your DDS if you ever need to recompile your green screen programs. You can archive your DDS.

* - *

Q: Does Presto require WebSmart, Clover, SmartCharts?

A: Presto does not require WebSmart, Clover or SmartCharts to run. However we do include SmartCharts with Presto at no charge. Presto is a standalone product. We would certainly encourage you to use WebSmart, Clover and SmartCharts to enhance your complete application modernization initiative.

* - *

Q: Does Presto handle RPG ILE in addition to OPM?

A: Presto works with RPG ILE or OPM. With Presto it does not matter what the underlying source is written in.

* - *

Q: Can we drop Seagull if we use Presto?

A: Yes.

* - *

Q: Do you send out updates for browser compatibility?

A: Yes.

* - *

Q: Which takes more bandwidth: green screen applications or Presto?

A: Web applications do take more network bandwidth than green screen apps, as a result of the mix of graphics and text.

* - *

Q: Regarding needing DDS, you still need to be able to compile your programs. That needs DDS. Your program still THINKS it has a screen. Doesn't it?

A: You are right, your programs will need the DDS to recompile, however, Presto does not use the DDS to present the program to the browser.

* - *

Q: Does the original DDS remain and can you continue to run the program as a green screen application?

A: Yes. Absolutely.

* - *

Q: Are there any deployment considerations that would impact the client PC?

A: No, not really. The browser needs to be compatible, that's all.

Appendix K - Web Page Design for Green Screeners

This chapter is aimed primarily at IBM i developers who have little or no experience writing web applications, and who want to start designing practical web pages for presenting data to end users. It will teach you the fundamentals of HTML and introduce you to some coding constructs that will enable you to consistently produce well-organized, attractive documents. We also discuss the issue of compatibility between various features of HTML and the available browsers.

Other Online Resources

This document is not intended to be an exhaustive treatment of the HTML language.

- For a good summary of current HTML vocabulary we recommend the W3 schools site: http://www.w3schools.com/html/default.asp.
- For current information on the latest HTML standard, or further official developments, visit http://www.w3.org/MarkUp/.

This last site also contains a good online tutorial on writing HTML so if you find this section boring, confusing, irrelevant or incomplete, then try out the Web resources instead - we won't be offended!



This document was originally published in conjunction with WebSmart, Fresche's CGI program and application development tool, and includes a number of references to how some of the HTML tags are used in WebSmart. For additional information on WebSmart, please visit our web site.

IBM i Screen Design vs. Web Page Design

On the IBM i, screen design is limited to 24 x 80 or 27 x 132 rows and columns. In web browsers, there are no such limitations. In addition, if you have used the Web to any extent, you will know that web pages can embed any kind of digital component: text, pictures, graphics, sound, movies, animation, etc. On the IBM i, you design screens by writing display file DDS, or using a design tool such as SDA (Screen Design Aid). Or, if you have ProGen Plus, its design tool, the PDA, takes care of constructing the screen layouts. Although the screens you produce with these tools are relatively inflexible in their layout, designing screens is a relatively simple

process. Part of the reason for this simplicity is due to the fact that you have limited choices in what you can do. In contrast, designing web pages can often be a daunting task because there are so many possibilities for how a page will appear. So the first step to success is to keep your pages simple. As you become more comfortable coding HTML, you can add more complexity to your pages.

HTML Development Tools

While display files are written in DDS on the IBM i, web pages are written in HTML. We'll explain HTML in detail later.

Using a design tool to build web pages is like using SDA to design green screens. Web page design tools have varying degrees of WYSIWYG layout capabilities, ranging from completely text-based to primarily visual component based, where the underlying HTML is mostly hidden from the designer. Available tools on the market (for Windows platforms) include, among many others:

1. **Adobe Dreamweaver.** Includes both a WYSIWIG visual design tool and a text-based editor.
2. **Namo Web Editor.** Another WYSIWIG editor.
3. **Eversoft 1st Page 2000 2.0.** This is a free text-based HTML editor available at www.evrsoft.com.
4. **HTMLKit.** A free text-based HTML editor, available at www.htmlkit.com. It includes an easy to use FTP client, if you need to upload/download pages remotely to your IBM i.
5. **Notepad.** No kidding! Experienced HTML coders prefer tools like Notepad because they have complete control over the HTML code.

The latest version of Dreamweaver includes extensive support for stylesheets (and encourages using stylesheets for layout wherever possible). We believe this is a positive direction, because stylesheets are easier to modify and maintain than HTML, and offer many more options for layout control than HTML.

Recent versions of Dreamweaver include extensive support for stylesheets (and encourages using stylesheets for layout wherever possible), which is a good approach to designing your site, offering many more options for layout control than HTML.

A tool for working with stylesheets exclusively is a product called TopStyle Lite, available in many places online. The full version of this product (TopStyle Pro) also purports to be a complete HTML text editor, although we find its interface somewhat less intuitive than other HTML-centric editors. Its strength remains in editing style sheets.

And there is a proliferation of other products out there. You should have no problem finding an editor to use.

To start with, in the sections below we will be coding HTML by hand so that you have a complete understanding of the syntax. Later on you can use the visual design tools to assist you.

Viewing the Source of Existing Pages

Another way to learn HTML is to view the source of pages you visit on the Web. Your browser should have a 'Source' option under the View menu. This will open a window with the source code for the page you are currently viewing. If the page uses frames, you may not see what you

expect, because pages with frames are handled differently than those without. If this happens, most browsers have a right-click option to open a frame in a new window, and if you do that, you can then view source on that frame section.

HTML: The Language that Isn't

HTML is the acronym for Hypertext Markup Language. The term 'language' only loosely applies to HTML. It is really more a set of a tag specifications than anything else. The term 'hyper' refers to the ability to jump from one place to another, using links. This is probably the single most important feature of HTML in terms of providing the power and ease of use of the Web. Unconstrained by green-screen design standards, HTML coding allows the web developer to take his audience anywhere with the click of a mouse button.

How to Create and Store HTML Files

HTML files are stored as ASCII text files. HTML files usually have an extension of .htm or .html (it doesn't matter which). On PCs, they can be stored in any Windows directory. On Unix systems, they are also stored in directories. On the IBM i, they can be stored in the IFS, which is essentially the same as a PC directory. You can write your pages on a PC, then FTP them to the IFS on your IBM i, or copy them there using Ops Nav (a component of Client Access). Another way to do this would be to edit a traditional IBM i source file, then copy it to the IFS, but this method won't provide you with the ability to do anything other than use SEU as a limited version of Notepad...

Testing and Debugging HTML

Because HTML is interpreted rather than compiled it is very easy to test your pages. Just find the files in Windows Explorer and double-click them to open them in your default web browser. If you coded everything correctly, it will display as you expect. If not, you may get strange results.

You can also get somewhat different results from different browsers. If all your pages are intended for Intranet use only, and you can control what browser everyone uses, you may not need to test other browsers. Otherwise, you should test all critical applications on at least IE 7 and 8, Firefox and Chrome.

Note that unlike compilers, browsers do not give you any kind of diagnostics when coding errors exist. This is where using an HTML design tool with built-in syntax checking can be helpful (you can run the WebSmart syntax checker by pressing F7 in the HTML view). As mentioned earlier, HTML is a tag-based language. Generally, each opening tag requires a corresponding closing tag. If you omit a closing tag, or place it in the wrong place, you won't get a diagnostic message from the browser. The page will still be rendered, but the results will be unpredictable.

Writing Your First Web Page

Let's start with the basic elements of an HTML page. We won't concern ourselves too much with describing the language syntax in detail. It should become self-evident as we work with it.

Here is a simple Hello, World! page:

```
<html>
<head>
<title>Hello, World Page</title>
</head>
<body>
Hello, World
</body>
</html>
```

You can open a new file in Notepad (don't use WordPad or Word, they will embed special characters in the file that will make it unreadable in a browser) and type in these lines (or cut and paste them, if you are viewing this document on your PC).

This is how the page looks in IE 8:

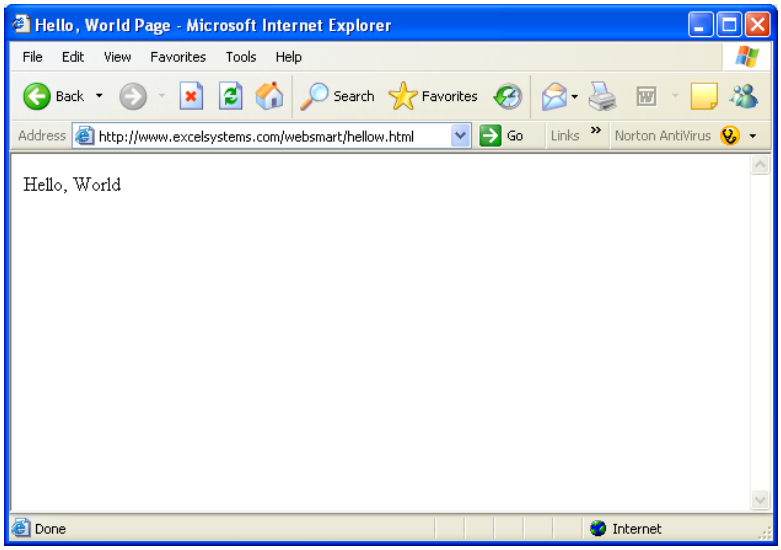


Figure 2-1. A simple 'Hello World' page.

To see this page for yourself, you can go to:
http://www.bcdsoftware.com/iseriessupport/documentation/websmart/htmlsamples/hellow.html

Explanation of the Code

<html>

Every page should begin with this tag. This informs the browser that it is about to render a page consisting of HTML. Note that the last element of the document is `</html>`.

This demonstrates the principle that (almost) every HTML tag has a corresponding end tag. The general syntax is: `<tagname> </tagname>` (where 'tagname' is any valid tag name). You can place anything between tags, including other tag pairs. This allows any level of indention of code,

which is one of the features of HTML that provides great flexibility in page construction. You can imagine the code to be indented like this:

```
<html>
                <head>
                        <title>
                                Hello, World Page
                        </title>
                </head>
                <body>
                        Hello, World
                </body>
</html>
```

Many design tools will do this to your code automatically. However, the indention levels can get pretty deep sometimes, so this is not always the easiest presentation style to read. Be aware that it really doesn't matter how you lay out the HTML. In fact, embedded blanks get ignored by the browser. To prove this, change your hellow.html document to look like this:

```
<html>
<head>
<title>Hello, World Page</title>
</head>
<body>
Hello,           World
</body>
</html>
```

Here we have added about 10 spaces between 'Hello' and 'World' inside the body tag. Now save the document, and view it in your browser (you can keep your browser open from the last time, and just click the Refresh button in IE, or the Reload button in Netscape. This causes the browser to think it has a new copy of the file, and to read and render all the HTML from scratch.)

Notice that the page appears exactly as before! Any blanks after the initial blank are ignored by the browser. In most cases, line breaks will also be ignored in the same way.

In most cases, line breaks are also ignored in the same way.



If you want to add more blank spaces between text, you must use the special value   The '&' character indicates a special set of characters are to follow; 'nbsp' stands for 'non-breaking space' and the semi-colon marks the end of the special character string. A little complicated, but after you use it a few times, you'll just memorize it, without remembering the logic behind it!

Try adding some spaces that will get rendered. Type this line between the body tags:

```
Hello,     World
```

The page should now look something like this:

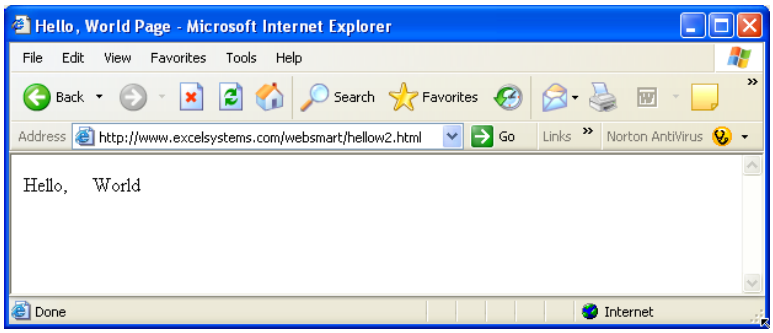


Figure 2-2. Adding non-breaking spaces to a web page.

To see this, try http://www.bcdsoftware.com/iseriessupport/documentation/websmart/htmlsamples/hellow2.html.

Okay - now to explain the rest of this document.

As a refresher, remember that the original code in hellow.html is:

```
<html>
<head>
<title>Hello, World Page</title>
</head>
<body>
Hello, World
</body>
</html>
```

Let's look at the rest of the tags.

<head>

The head tag is where you type HTML that is processed before the body of the page. Although our example contains just the page title, it could also contain meta tags, JavaScript or cascading style sheets (all explained in later sections). The general format of this tag is `<head>....</head>`.

By now you've probably got the hang of the `<tag> </tag>` notion, so we won't mention it again unless it is particularly relevant.

<title></title>

The title tag pair contains text that appears in the top border of the browser window. In IE, the text is: Hello, World Page - Microsoft Internet Explorer.

Page titles are part of the criteria that search engines use when evaluating and indexing your page. So if you would like your site to get a better search engine ranking and more visits, it's worth taking a moment to add a meaningful page title.

<body></body>

The body pair is used to contain the main content of the page (the body of the page). As mentioned before, it can contain text, HTML tags, or a combination of the two. In our example, it contains the simple text: Hello, World.

To summarize: Every document you create should generally contain at least this much HTML:

```
<html>
<head>
```

```
<title> </title>
</head>
<body>
</body>
</html>
```

However, browsers are getting smarter at dealing with structural anomalies in the HTML, so you may sometimes get away with missing some of these.

Formatting Text - Simple Tags

One of the first things you will want to do to your page is to add formatting. There are several pre-defined tags that you can use to format text. The pre-defined formatting tags are: H1, H2, H3, H4, H5 and H6.



You can specify tags in either upper or lower case, or a mix of the two, with no difference in outcome. While most people find lower case easier to read, it is strictly a matter of preference as to which you use.

Each one of these tags formats text in a different size of font, with H1 being the largest. Try this sample page:

```
<html>
<head>
<title>Different Formats for Text</title>
</head>
<body>
<H1>This is H1 Formatted Text</H1>
<H2>This is H2 Formatted Text</H2>
<H3>This is H3 Formatted Text</H3>
<H4>This is H4 Formatted Text</H4>
<H5>This is H5 Formatted Text</H5>
<H6>This is H6 Formatted Text</H6>
This is normal text.
</body></html>
```

The resulting page should look like this:

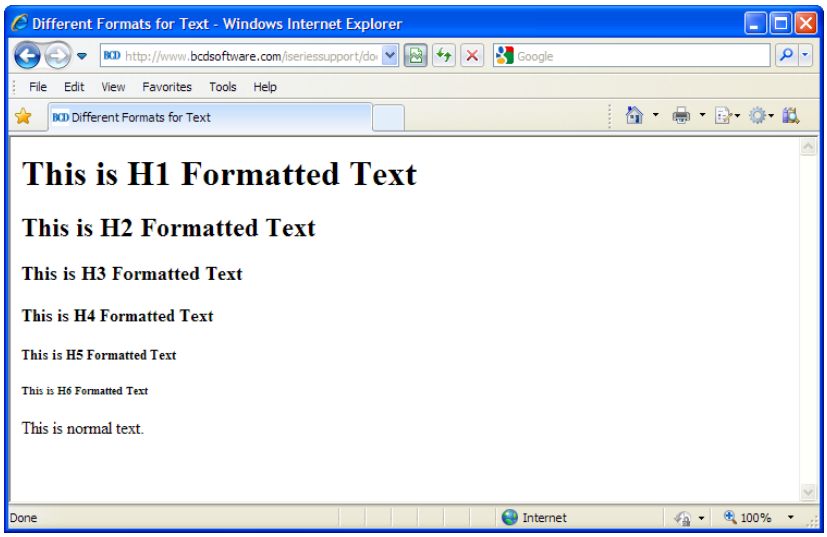


Figure 2-3. An illustration of the standard formatting for HTML headers 1 through 6.

You can apply additional effects to these formats. Some of these are keywords that go inside the H tags, while some are separate tags that can be used in conjunction with these, or by themselves. A keyword has the general format: `KEYWORD=VALUE`. Keywords are like parameters for tags. Occasionally a keyword has no value.

For example,

```
<H1 ALIGN=CENTER>This is H1 Text</H1>
```

is valid, as is:

```
<CENTER>This is normal text</CENTER>
```

In the first example, ALIGN=CENTER is a keyword/value pair. It does not require a corresponding end pair, as does a tag- demonstrated in the second example.



A better technique than specifying one or more keywords on your tags is to set up style sheets that override the appearance of every instance of a particular tag. Style sheets are discussed later in this chapter.

Here's another example:

```
<html>
<head>
<title>Different Formats for Text</title>
</head>
<body>
<H1 align=center>This is H1 Formatted Text</H1>
<H2 align=left>This is H2 Formatted Text</H2>
<H3 align=right>This is H3 Formatted Text</H3>
<center>This is normal text.</center>
</body>
</html>
```

This sets a different alignment option for each line.

The rendered page should look like this:

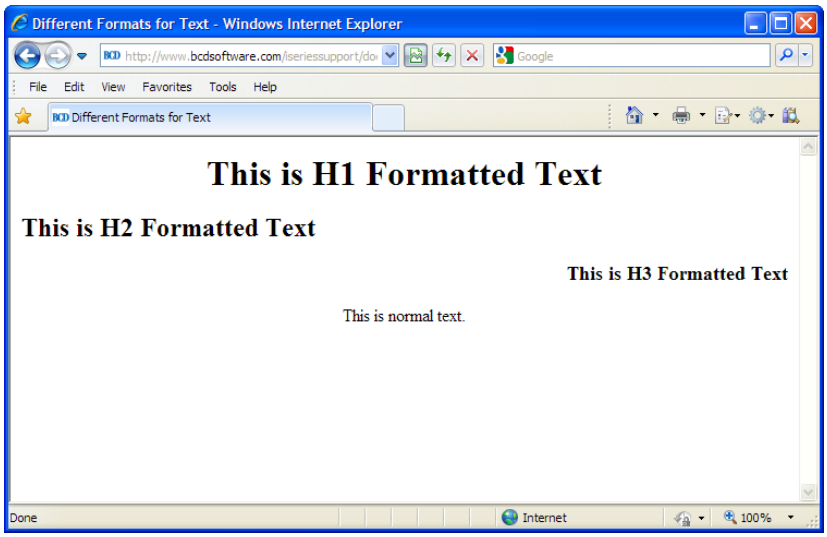


Figure 2-4. An illustration of the available HTML alignment options.

You can try this page at http://www.bcdsoftware.com/iseriessupport/documentation/websmart/htmlsamples/keywdsandtags.html.

Other commonly used formatting tags are:

- </strong - usually bolds text.
- - bold
- - italics

The Paragraph and the Pre-formatted Tags

The paragraph tag pair is `<P></P>`. This is one of the few tags where the `</P>` can be omitted with no unpredictable results arising (another common one is `</li>` - end list item). The `<P>` marks the start of a new paragraph. Within a paragraph, text will wrap within the browser until the next `<P>` or some other tag is encountered.

The `<pre></pre>` tag has almost the exact opposite effect. Any text inside this tag pair is displayed exactly as stored in the source file for the page, including any embedded blanks (remember, normally, all embedded blanks but one between words are ignored by the browser). The `<pre></pre>` tag pair is useful for displaying IBM i spool files. Because it keeps the embedded blanks, you are guaranteed that column headings will line up with data in the page. Later, we explain HTML tables, which are a much better way of presenting tabular information. However, with two simple tags, the `<pre></pre>` pair can help you deliver IBM i static data in HTML forms with minimal effort.

Using Different Fonts for Text

The default font is a serif font ('serif' is the curly bit on letters like T, F, etc.). This text is Minion Pro, which is a serif font. Arial is a sans (French- "without") serif font. Many of the more popular and up-to-date sites use sans serif fonts to enhance their appearance, because sans serif fonts tend to look cleaner on a screen.

There are two basic methods for setting the type face for text. One is to use the `<font> </font>` tag pair, and the other is to use cascading style sheets. Most current browsers support cascading style sheets, but on occasion you may find it useful to use `<font> </font>` pairs to set the type

face of text. Be aware, though, that the   tag pair is being 'deprecated' by W3C, the Web Consortium responsible for the HTML language standard. This means they are hoping to phase out its use, and replace it with Cascading Style Sheets exclusively. This probably won't happen for at least a few more years, since it takes the web user community a while to completely upgrade to the latest browsers.

To set the font, you can do something like this:

```
<font face="Arial">text goes here</font>
```

or:

```
<font face="helvetica, arial, ms sans serif">text goes here</font>
```

The first example will set the font to Arial, if it is supported by the browser.

The second example will attempt to use Helvetica font first. If that is not available, the browser will attempt to use Arial, and so on. You should make a practice of always closing tags with tags, without allowing any mismatched pairs in between. When we look at constructing tables later on, this becomes an important consideration. We will come back to this issue at that time.

Changing Font Sizes

You can change the size of text by including the SIZE keyword inside tag pairs, as follows:

```
<font size=2>Some text</font>
```

This uses an absolute size. Valid values are 1 to 7.

```
<font size=-1>Some smaller sized text</font>
```

This uses a relative size: 1 smaller than the size of font for the previous text.

```
<font size=+1>Some larger sized text</font>
```

This uses a relative size that is 1 larger than the size of font for the previous text.

Placing Images on Pages

You can place images on pages by using the <IMG> tag. This tag breaks the rule of using a corresponding </TAGNAME> tag to complete the code. That's because it generally takes the place of plain text, so a single tag suffices to express the object. This example also sets the width and height of the image, in pixels:

```
<img src="img_id/football.gif" width=4 height=100>
```

This next example will show the text ‘Football’ if the image is not available:

```
<img src="img_id/football.gif" width=4 height=100 alt="Football">
```

This example places a 3-pixel wide border around the image:

```
<img src="img_id/football.gif" width=4 height=100 border="3">
```

You can use alignment keywords within the <img> tag. The alignment options are:

```
ALIGN=TOP | BOTTOM | MIDDLE | LEFT | RIGHT | TEXTTOP | ABSMIDDLE | BASELINE | ABSBOTTOM
```

In order to precisely control the placement of images on a page, you will probably need to construct tables, and include <IMG> tags inside the tag pairs for table element definitions - <TD></TD>. See the section below on tables for details.

Using Color in Your Pages

Color can be applied to plain text, links (visited, not-yet-visited, or being-clicked), or to backgrounds - either the entire page (inside the body tag) or in table elements.

Here is an example that sets the font color for some text to white (invisible, unless the background color is changed).

```
<font color="#FFFFFF">This is white text</font>
```

The characters "#FFFFFF" represent a hexadecimal value for a color. The pound sign (#) indicates the start of a hex value. The possible range of values is from 000000 to FFFFFF (using the numbers 0-9 and letters A-F). The order of the hex components is Red, Green, Blue. In other words, whatever change you make to the first 2 characters affects red's contribution to the overall color, etc. This yields 255 * 255 *255, or 16,581,375 colors. However, in reality, the number you should select from is considerably fewer, and there is also the option to use named colors instead of hex values.

The following is an excerpt from http://www.w3.org/MarkUp/Guide/Style.html#hexcolors by Dave Raggett (http://www.w3.org/People/Raggett/).

Named colors

The standard set of color names is: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow. These 16 colors are defined in HTML 3.2 and 4.01 and correspond to the basic VGA set on PCs. Most browsers accept a wider set of color names but use of these is not recommended.

Color names and RGB values

Here are the RGB values for the named colors:

- black = "#000000"
- maroon = "#800000"
- green = "#008000"
- navy = "#000080"
- silver = "#C0C0C0"
- red = "#FF0000"
- lime = "#00FF00"
- blue = "#0000FF"
- gray = "#808080"
- purple = "#800080"
- olive = "#808000"
- teal = "#008080"
- white = "#FFFFFF"
- fuchsia = "#FF00FF"
- yellow = "#FFFF00"
- aqua = "#00FFFF"

Thus, the color value "#800080" is the same as "purple".

Examples of Setting Colors for Backgrounds and Text

The background color for an entire page can be set by using the BGCOLOR keyword inside the body tag, as follows:

```
<body bgcolor="aqua"> Black text on an aqua background</body>
```

This example also sets the color of the text for the entire page:

```
<body bgcolor="aqua" text="purple">Purple text on aqua background
</body>
```

This sets the color of the first line of text to purple, while the next line is black:

```
<body bgcolor="aqua" text="purple">Purple text on aqua background
<P>
<font color="black">Black text on aqua background</font>
<P>
This text is also Purple.
</body>
```

This demonstrates the principle that higher level tags can be progressively overridden by subsequent ones, yet the browser reverts to the higher level setting after encountering the closing tag for the subsequent one:

You can also set the colors for hypertext links. There are three attributes for this:

- link for unvisited links
- vlink for visited links
- alink for the color used when you click the link

Here is an example that sets all three:

```
<body bgcolor="white" text="black" link="navy" vlink="maroon" alink="red">
```

This next example sets the background color for an entire row in a table to a light gray. We haven't presented the complete coding for tables yet, but this will give you an idea:

```
<tr bgcolor="#cccccc"><td>Table column</td></tr>
```

Navigating Using Links

The most common tool for navigating within or between pages is with links. A link can point to another location within the same page, another page on the same site, or a completely different URL. The reference can be absolute (e.g. http://www.cnn.com) or relative to the current page (e.g. /htmlguide/barebones.html).

Here are some example links:

```
<a href="http://www.cnn.com">CNN</a>
```

In this link, the text CNN goes to the external site CNN.com.



The Web standard for hyperlink text has been underlined blue text, but more recently site designers will often make links match the site's color scheme.

This link goes to a page called mypage.html within the current domain:

```
<a href="mypage.html">My home page</a>
```

So if I am currently at www.bigcompany.com/index.html, the browser will attempt to go to www.bigcompany.com/mypage.html.

This is a mail-to link:

```
<a href="mailto:excel@www.excelsystems.com">Software Support</a>
```

When clicked, it will pop up the default mail program associated with the browser you are using, with a new message started and the address above as the To: address.

This is an example of using an image as a link:

```
<a href="/"><img src="logo.gif" alt="home page"></a>
```

The image file is called logo.gif, stored in the same directory as the current page. Hovering the mouse over the image will cause the text "home page" to appear. Also, if the image cannot be loaded (the name or directory is incorrect) then the text "home page" will appear in its place. The href="/" directs the link to the root directory and page as defined in the web server for the current site.

This image links uses an absolute URL to go to another site:

```
<a href="http://www.bcdsoftware.com><img src="img_id/bcd_logo.gif" width=168
height=100></a>
```

It also uses the width and height keywords to determine the dimensions of the image on the page, regardless of its actual size. These dimensions are in pixels.

Defining Tables

Tables are one of the most useful HTML elements. They help you do the following:

- Control the precise location of text and/or graphics on a page.
- Control the background image or text for areas of a page.
- Align data in columns.
- Create newspaper or magazine-style layouts.
- Avoid the need to use frames.

You will often find that you need to nest tables within tables to get the desired effect. Unless you structure the HTML of you page well, nested tables can be difficult to manage in the raw source code, because it is hard to track the nesting of the various tag pairs. The WYSWIG HTML editors can sometimes help in this regard, but only to a certain extent. Creating new table rows or columns, especially inside nested tables, is often a hit-and-miss proposition when using a design tool. You will probably find yourself resorting to working with the raw source code whether you want to or not.

Here is the code for a 2 x 2 table (2 rows, 2 columns):

```
<table border=1 width=100%>
<tr><td>Col 1,Row 1</td><td>Col 2,Row 1</td></tr>
<tr><td>Col 1,Row 2</td><td>Col 2,Row 2</td></tr>
</table>
```

Here's what it looks like in the browser:

Col 1,Row 1	Col 2,Row 1
Col 1,Row 2	Col 2,Row 2

Figure 2-5. This is how a simple table with two rows and two columns is rendered in your browser.

You can find this table at http://www.bcdsoftware.com/iseriessupport/documentation/websmart/htmlsamples/Table2x2.html.



To save space, from now on I'll omit the usual surrounding `<HTML>`, `<BODY>` tags, etc. from our examples. To see the full source of these pages, you can use your browser's View Source option.

The `<table></table>` tag pair defines the overall characteristics of the table. The table above has the default attributes of a 1-pixel border around all the cells and a width spanning 100% of the browser window. The most important `TABLE` keywords are: WIDTH (in pixels, or a percentage), BORDER, CELLSPACING and CELLPADDING (all in pixels).

The `<TR></TR>` tag pair defines the contents of the row. In the table above, Each row contains two `<TD></TD>` tag pairs, corresponding to 2 columns in each row. You may wonder why you need to define the number of columns in each row, rather than defining that value once per table. The

answer is that differing rows can have differing number of columns. It is possible to have a cell span 2 or more columns. Here is an example of this:

```
<table border=1 width=100%>
<tr><td colspan=2>Cols 1 and 2,Row 1</td></tr>
<tr><td>Col 1,Row 2</td><td>Col 2,Row 2</td></tr>
</table>
```

Here is an illustration of this table (posted at http://www.bcdsoftware.com/iseriessupport/documentation/websmart/htmlsamples/Table1+2.html).

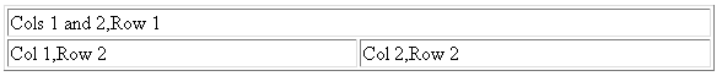


Figure 2-6. An example of a row that spans more than one column.

And just as you can use the COLSPAN keyword to have a row span multiple columns, you can use the ROWSPAN keyword to have a cell span more than one row.

You can use the BGCOLOR keyword inside the <TR> tag to set the background color for the entire row. You can also specify the alignment of the cells inside the row, using the ALIGN keyword. Valid values are: LEFT, RIGHT, CENTER, BOTTOM. The VALIGN keyword defines vertical alignment of the cells. Valid values are: TOP, BOTTOM, MIDDLE.

The <TD></TD> ("table" definition) defines the properties of a single cell within a table. It must appear inside the <TR></TR> tag pair. Valid keywords are:

```
BGCOLOR = color
ALIGN = LEFT, RIGHT, CENTER.
VALIGN = TOP, BOTTOM, MIDDLE
COLSPAN = Pixels
ROWSPAN = Pixels
NOWRAP
WIDTH = Pixels or %
```

Here's an example that uses a number of these features:



Figure 2-7. An illustration of using background colors in your table cells.

You can find this online at http://www.bcdsoftware.com/iseriessupport/documentation/websmart/htmlsamples/TableFeat1.html.

The HTML for this table looks like this:

```
<table border=0 width=90% cellspacing=1>
        <tr>
                <td colspan=2 bgcolor="gray">Cols  1 and 2,Row 1  </td>
        </tr>
        <tr bgcolor="aqua">
                <td width=70%>Col 1,Row 2</td>
                <td width=30%>Col 2,Row 2</td>
        </tr>
</table>
```

Indenting small tables does make them easier to read.

About the WIDTH Attribute

Besides being a good demonstration of how to misuse color, the page above demonstrates some interesting implications about the WIDTH attribute.

Notice that, at the Table level, we specified a width of 90%. This left a white space of 10% to the right of the table. No matter how big or small you resize the browser window, this white space will remain. So, at the table level, the width attribute is relative to the window (or frame, or other table) in which it is built).

The WIDTH attributes at the cell level are relative to the table. So, 70% represents 70% of the table's width, which is 90% of the browser window's width. The next cell is as wide as 30% of the table's width.

You can build page layouts much like magazine layouts by using these attributes correctly.

Here is the source for a page layout with a heading table, followed by a left side table and a right side:

```
<table border="0" width="90%">
        <tr bgcolor="D3D3D3">
        <td size="-1" align="center">
                <font face="Arial">            This is the top row in the outer table
                </font>
        </td></tr>
        <tr><td>
          <table border=2 width=100% cellspacing=1>
            <tr><td width="30%" rowspan="7" valign="TOP"><table border="2"
width="100%"><tr><td>Left Hand Column</td></tr>
                        <tr><td>Another row in this column</td</tr>
                        </table></td>
            <td width="70%">
                <table border="1" width="100%">
                  <tr><td>Inside row 1</td></tr>
                  <tr><td>Inside row 2</td></tr>
                  <tr><td>Inside row 3</td></tr>
                  <tr><td>Inside row 4</td></tr>
                  <tr><td>Inside row 5</td></tr>
                  <tr><td>Inside row 6</td></tr>
                  <tr><td>Inside row 7</td></tr></table>
            </td></tr>
          </table>
        </td></tr>
</table>
```

The page is rendered like this:

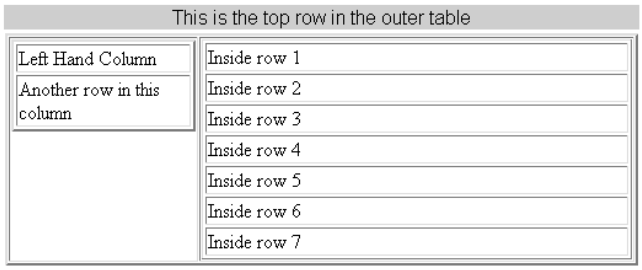


Figure 2-8. A more complex table, using several different table cell and row attributes.

You can see a live example of this table at http://www.bcdsoftware.com/iseriessupport/documentation/websmart/htmlsamples/NestTables1.html.

The tables work as follows:

- The first table is the outermost one.
- The first row uses the BGCOLOR attribute to set the color, and we used a tag inside the <TD> tag to set the font to Arial.
- Notice that the tag precedes the </TD> tag. This is usually the best way to ensure that fonts get rendered as expected within tables.

You can't have a font tag span more than one cell.

Controlling Column Widths

When a Web browser renders an HTML table it needs to determine how wide each column will be in the display area. By default these widths are calculated using the contents of the table data (td) tags, which define the columns within a table row.

This can be illustrated using a table with only one row as shown below:

```
<table border=1>
  <tr>
        <td>Lorem</td>
    <td>Lorem ipsum dolor sit</td>
    <td>Lorem</td>
    <td>Lorem ipsum dolor</td>
  </tr>
</table>
```

The rendered version of this code is shown below:

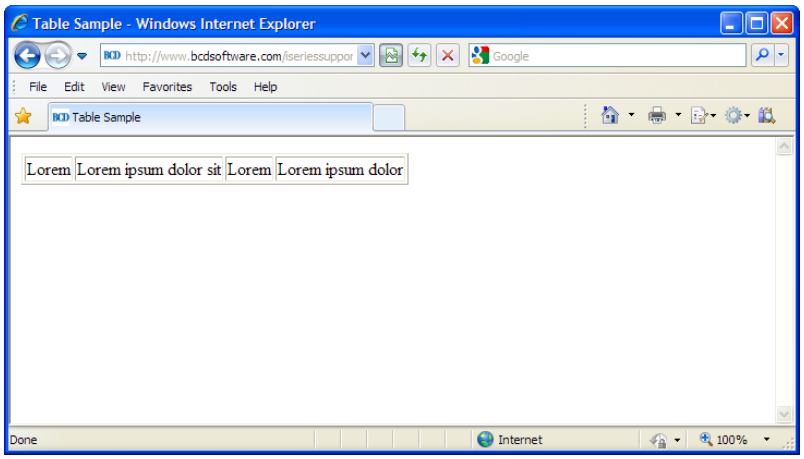


Figure 2-9. An illustration of a table with no column-width overrides.

Note that the column widths exactly match the width of the text they contain. You can control the width of the columns by defining a width for the table, in which case you can make the columns wider but they will still maintain their relative widths based on their contents.

You can also define a width for each column in pixels, or as a percentage of the display area, by entering a "width" attribute for each "td" tag. However, the browsers do not treat this attribute consistently, and it can be difficult to achieve the result that you are seeking. Internet Explorer

supports the "colgroup" tag, which allows you to define widths for each column in the table that follows; you can experiment with this tag.

A reliable and browser-independent method of spacing columns involves the insertion of a transparent graphic with a consistent width and height in each column. This is illustrated in the following code:

```
<table border=1>
  <tr>
    <td><img src="trans.gif" width=150 height=1><br>Lorem</td>
    <td><img src="trans.gif" width=150 height=1><br>Lorem ipsum dolor sit</td>
    <td><img src="trans.gif" width=150 height=1><br>Lorem</td>
    <td><img src="trans.gif" width=150 height=1><br>Lorem ipsum dolor</td>
  </tr>
</table>
```

The rendered version of this code is shown below:

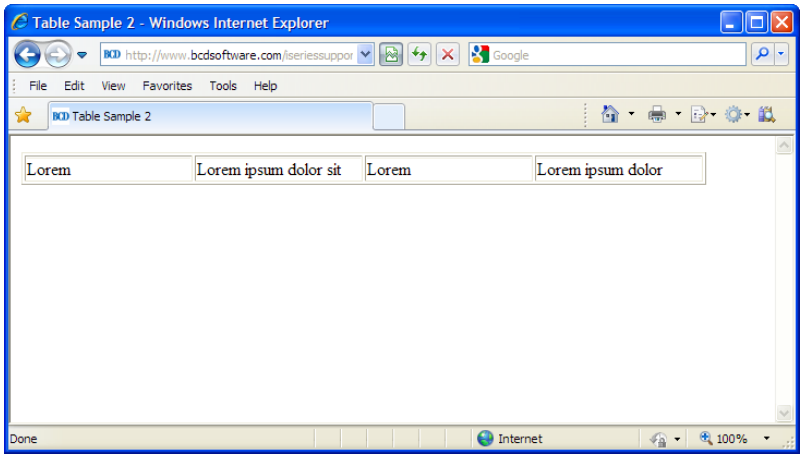


Figure 2-10. Overriding table column widths.

In this case the transparent GIF image becomes the widest single component in the column, so the browser bases the column width on the width of the image. Because this width is consistent for each column in this example, the columns are rendered with even spacing. If you do not require even spacing, you can format the widths of each column by varying the width of the transparent image. If the table contains multiple rows, the columns in subsequent rows will inherit their widths from the cells containing the graphics. You only need to insert the graphic in one table cell for each column. A good way of achieving this is to place the graphic in the cells containing the column headings.

If a cell contains a single component, such as a long text string, a table or another graphic, that exceeds the width of the transparent image, the browser will adjust the column width accordingly. Text strings are a special case because the text within a column will wrap automatically. You can force the text to wrap and maintain your chosen column widths by setting the width of the table in pixels using the "width" attribute. When choosing the table width you should take into account the combined widths of the columns, the cell spacing, cell padding and border width. There are so many variables that it normally takes some trial and error before a satisfactory result is achieved.

Interactive Pages - Using Forms and Forms Controls

Forms are used to create interactive web pages. These are pages that include items such as submit buttons, radio buttons, drop-down boxes, input fields, input text boxes, etc. If your application requires any sort of user input, or needs to provide conditional branching to other pages, then you will probably need to use forms.

Interactive pages are generally used in conjunction with server side CGI programs or Java servlets. For example, WebSmart programs can receive input from forms as parameters. This section includes a general reference for forms.

Below is a summary of the various components available for forms, along with their corresponding HTML tags.

Forms

<FORM> </FORM>

Denote the boundaries for a form. All tags for controls contained within these tags belong to the form.



'Controls' refers to the input boxes, checkboxes, etc. that are contained within the form. They are normally defined with an <input...> tag.

You can have multiple forms on a page. Note that a form is not actually visible to the user, although the controls it contains are. The data sent to the web server from the page depends on which form contains the submit button pressed by the user. No data is sent from any other form, no matter how many are on the page. This is explained in more detail later in this chapter, when we discuss the `SUBMIT` tag.

Here is some sample code for a form with a single submit button (you can try running this page yourself at http://www.bcdsoftware.com/iseriessupport/documentation/websmart/htmlsamples/myform.html).

```
<html>
<head>  <title>Example Form to Call MYPGM.PGM</title></head>
<body>
<form action="mypgm.pgm" method="GET">
<p>Click the submit button to attempt to call mypgm.pgm</p>
<p><input type="SUBMIT" name="Submit" value="Submit">
</form></body></html>
```

When the user clicks the submit button, the browser sends a message to the Web Server to invoke program mypgm. In this case, if you go to that page and click the submit button you should get a message indicating the page cannot be found. This is because program 'mypgm' does not exist - this just shows you you how you can call programs with forms.

Text Input Fields

<INPUT TYPE=TEXT>

Presents a text control that allows the user to type characters as input. This input is returned as data to the server. This tag has attributes that allow you to define the size of the text box, and the maximum size of data it can contain. Here's an example tag:

```
<input type="TEXT" name="CUSTNAME" size="20" maxlength="30">
```

This defines an input field named CUSTNAME, with a width that displays up to 20 characters, and allows a total of 30 characters to be typed in. The NAME attribute is supplied as a parameter name to the server when the user submits the form containing this control. For example, if the user types "John Q Public" in the text box, the browser will send this data to the server:

```
&CUSTNAME=John%20Q%20Public.
```

The '&' denotes the start of a parameter name. WebSmart programs can get the data associated with any parameter name. In this case, the program would retrieve the value John Q Public (although the browser sends %20 for blanks, these are correctly interpreted by WebSmart programs).

You can also use the VALUE attribute with this tag to display an existing value in the text box. For example, if you are writing a file maintenance program with WebSmart, you can populate the text box with the current contents of a database field. The user sees this value, and can then change it, if desired. Here's an example using the value keyword:

```
<input type="TEXT" name="TODAY" value="2008/06/14">
```

When the page is displayed, the user sees the date in the box, and can type over it.

<INPUT TYPE=PASSWORD>

This tag is similar to <INPUT TYPE=TEXT>, except any characters typed by the user appear as asterisks. Use this tag for entry of passwords or data that the user wants to hide from others. Another difference between TYPE=TEXT and TYPE=PASSWORD is apparent when a page containing both of these tags is redisplayed (for example, by using the back button on the browser). Text fields retain their contents, while password fields are cleared.

<INPUT TYPE=HIDDEN>

This tag is used to store a hidden value on a page, such as a parameter that needs to be shared between pages. The tag does not have any visible effect on the page. It is only useful when used with the VALUE keyword. Here is an example:

```
<input type="HIDDEN" name="TASK" value="CHANGE">
```

When you submit a form with this tag, the browser sends this string to the server:

```
&TASK=CHANGE
```

WebSmart programs use the TASK parameter to control the logic flow of the program.

<SELECT> </SELECT>

This tag pair defines the properties for a drop down box, showing one or more list items in the box. Inside the tag pair, you specify <OPTION></OPTION> tag pairs, to define each item in the list. Here is an example that presents a drop down box allowing the user to select one of three colors:

```
<select name="COLOR">
        <option selected value="R">Red</option>
        <option value="G">Green</option>
        <option value="B">Blue</option>
</select>
```

The SELECTED attribute causes the word "Red" to appear as the default entry that appears to the user. The VALUE attribute is the value of the parameter returned by the browser. For example, if the user selects Green, the following parameter name/value pair is sent to the server:

```
&COLOR=G
```

By default, the select box shows one item. You can have more than one item appear by using the SIZE attribute. For example,

```
<select name="COLOR" size="3">
```

would cause all three colors to appear in a list. You can also allow the user to select multiple items at once, using the MULTIPLE attribute, as follows:

```
<select name="COLOR" size="3" multiple>
```

If the user selects more than one color (by pressing the ctrl key at the same time as clicking) then a parameter name/value pair is returned for each selection. For example if the user selects colors Red and Blue this string is returned:

```
&COLOR=RED&COLOR=BLUE
```

In WebSmart, you can use the GETNXTPRM function to get the next available parameter value for repeating parameter names.

<INPUT TYPE=RADIO>

This tag is used to place a radio button on the page. You can group radio buttons so that the choice made by the user is mutually exclusive of the other choices. You do this by specifying the same value for the NAME attribute. For example, the following code shows a row of radio buttons, with text to the right of each one:

```
<input type="RADIO" name="Choice" value="Change">Change
<input type="RADIO" name="Choice" value="Display">Display
<input type="RADIO" name="Choice" value="Delete">Delete
```

The user can activate only one button in the group when clicking on it. If the user clicks on Change, then the following parameter name/value pair is returned:

```
&Choice=Change
```

You can pre-set one of the buttons in the group as selected, by using the CHECKED attribute. For example, this input:

```
<input type="RADIO" name="Choice" value="Display" checked>
```

causes the Display button in the above group to be pre-selected. It will appear to the user with a black center, instead of a white one.

<INPUT TYPE="CHECKBOX">

This tag presents a checkbox. Unlike the RADIO control, you cannot use it to force the user to make one choice from several. The best use is in Yes/No situations. For example:

```
Click here:<input type="CHECKBOX" name="EMAIL" value="YES"> to receive an email.
```

Note that a value for this parameter is only returned if the box is checked, either by the user clicking on it, or by specifying the CHECKED attribute for it. For example:

```
Click here:<input type="CHECKBOX" name="EMAIL" value="YES" checked> to receive email.
```

This line will cause the check box to be pre-checked when the page is presented. This is a useful way of establishing defaults for values.

<INPUT TYPE="SUBMIT">

Use this tag to create a Submit button. For example:

```
<input type="SUBMIT" name="OKClick" value="OK">
```

Code that Affects the Entire Page

The basic structure of an HTML document consists of a head section and a body section. So far, we have focused mainly on the information contained in the body of a document. There are instructions that apply to the entire document, however, and these go inside the head section. These range from JavaScript tags and code to style sheets and meta tags.

JavaScript

When you run a WebSmart program, the Web pages produced are dynamically created by combining the HTML code in your definition with data from the server. This process is not limited to the visible content of the page, as it can also define the outcome of clicking on buttons, hyperlinks and other components on the page. This is accomplished using an executable CGI program, which resides on the IBM i.

Another way in which the appearance and behavior of a Web page can be customized is by using JavaScript. JavaScript is an object-oriented programming language that runs within the Web browser. This means that it is a client side programming language as opposed to the server side approach taken by CGI programs. JavaScript programs are defined in one of two ways:

1. **Inline.** The scripts are placed at a specific location in the HTML code, and run when the code is loaded into the browser. This can be within the head or the body of the HTML document.
2. **Using Function Definitions.** Functions can be defined that are executed as a result of a browser event, such as a mouse click, or an explicit inline call from outside the function. Although these can be inserted into the body of the document, it is safer to locate them in the head portion. This ensures that they have already been loaded before an event is triggered by which they are executed.

Here is an example:

```
<script type="text/javascript">
today = new Date()
document.write('This data snapshot taken on : ',today.toLocaleString())
</script>
```

This inline snippet displays the date and time at which the page was loaded into the browser.

It illustrates two components that should be present for each script:

1. The opening script tag - <script type="text/javascript">
2. The closing script tag - </script>

This example produces HTML text that will appear on the page using the document.write() method. The text will appear at the location where the inline script is placed.

This script illustrates the object-oriented nature of JavaScript. Each Web page displayed in a browser is actually a visual representation of the Document Object Model (DOM) object. This is an object which is accessed using the name "document". The data and methods that it contains are accessed by appending a period and the name of the method or data structure after the string "document", in this case ".write(<string>)".

The following script illustrates the use of a JavaScript function definition.

```
<script type="text/javascript">
function MM_preloadImages() {
  var d=document; if(d.images){ if(!d.MM_p) d.MM_p=new Array();
    var i,j=d.MM_p.length,a=MM_preloadImages.arguments; for(i=0; i<a.length; i++)
    if (a[i].indexOf("#")!=0){ d.MM_p[j]=new Image; d.MM_p[j++].src=a[i];}}
}
MM_preloadImages('/enter image arguments here')
</script>
```

The script defines the function MM_preloadImages() and executes it. Because the function and the call to the function are together, this snippet can be inserted into the head or the body of an HTML document as long as no other calls to the function are added. Coding this way ensures that the function is always loaded before it is executed. This function also uses the DOM object. The statement "var d=document;" assigns the DOM to the variable "d" so that less typing is required to reference it.

There are many excellent online resources for learning JavaScript.

Style Sheets

Early Web browsers provided very little control over the appearance of text on a Web Page. Tags like <B> (bold) and <I> (italic) were available, but other attributes such as font color, size and typeface could not be controlled. The first HTML tag that provided control over many text attributes was the font tag. The disadvantage with this approach was that "begin" and "end" font tags were required for each change in text attribute. This resulted in many extra tags being distributed throughout the document. The problem is especially acute when setting font attributes within table cells, as the font attributes only apply to text within a single cell. In order to set the typeface for an entire table, for example, an opening and closing font tag has to be added to each cell. Depending on the content of the cells and the complexity of the text attributes defined in each font tag, this could easily triple the size (in bytes) of the document.

Style sheets provide a much more economical method of setting text attributes. The typeface, color, weight and many other text attributes can be associated with a type of tag, e.g. the TD tag that defines each cell of a table, or a user-defined class name. The simple style sheet shown below is used by the WebSmart templates:

```
<style type="text/css">
TD {font-family:Arial; font-size:10pt}
</style>
```

Like the JavaScript examples in the previous section, a style sheet has opening and closing tags, and the contents of the style sheet are contained within an HTML comment to hide them from browsers that do not support style sheets. Instead of a "Language" attribute, however, the opening "style" tag has a "type" attribute. This indicates the type of style sheet language being used. In this case it is "text/css". CSS is short form for Cascading Style Sheet.

The style above simply instructs the browser to use the Arial type face, with a font size of 10pt, for all text contained between opening and closing TD tags. This avoids the need to use the Font tag at all, and it accomplishes the task using a very condensed code segment.

As mentioned above, style sheets can also define class names as illustrated below:

```
<style type="text/css">
.important {background: yellow}
</style>
```

The class name in this case is "important". The browser differentiates between a tag type and a class name by checking for a period before the label. In this case "important" is preceded by a period indicating that it is a class name.

To set the style of a piece of text to "important" you could use the following approach:

```
This is normal text. <span class="important">This is important.</span>
```

The SPAN tag is used to group inline text so that scripting or style rules can be applied to the content. Other tags, such as the paragraph (<P>) tag can also use the CLASS attribute, although this would assign the style to the entire paragraph rather than a few words. Currently, style sheet interpretation by browsers is improving, and it is generally a good idea to use style sheets to consistently format your pages. For more information on Style Sheets please refer to the following URL:

http://www.w3.org/Style/CSS/

This is the official World Wide Web Consortium (W3C) cascading style sheet page. W3C is the organization that oversees the creation and adoption of World Wide Web standards, so this is the best starting point for your research. Although much of the information presented on this site is technical in nature, it contains a number of resources for beginners.

Meta Tags

The term META, in this context, is an abbreviation of Meta-data. Meta-data is data about data. In terms of an HTML document, META tags provide information about the document, which is in itself data, and how it should be processed. All HTML documents created by WebSmart programs that were derived from a WebSmart template will contain at least one META tag as shown below:

```
<meta http-equiv="Pragma" content="no-cache">
```

This tag tells the Web browser that the document should not be stored in its cache. The cache is an area in memory and on disk where Web documents of various types are stored so that they are not downloaded from the remote site each time the page is accessed. Caching speeds up the performance of a Web browser because it does not have to wait for a document to reload if you have already viewed it once. This is especially useful in the case of downloaded graphics, as they tend to be much larger than the text of an HTML file.

If the pages you are viewing are produced by a WebSmart program, however, you should assume that the page changes each time you view it, and therefore it should not be cached. The "no-cache" META tag prevents the caching of all WebSmart pages, so you are guaranteed to see the latest version. All META tags must be present in the HEAD section of an HTML document. There are actually several dozen types of META tag. Other common examples include the following:

```
<meta name="description" content="Description of your page">
<meta name="keywords" content="list of keywords">
```

These two tags are used by search engines when they are indexing your pages, and they should only be used if you want a search engine, such as Google, to list your pages when a search is

performed by another user. If your pages include the "description" tag, the string you enter for the "content" attribute is used to identify your page on the search engine's website. If your pages include the "keywords" tag, the string you enter for the "content" attribute defines a set of words that are stored by the search engine, and used to match the search term entered by the user.

If you are a bookseller, for example, you might include the following content string "book, books, literature, writers, authors". If a search word entered by a user matched one of the words in this list, your page would appear in the search results.

As always, the best source of information about Meta tags in general is the Web itself.

CSS (Cascading Style Sheets)

This section will teach you the fundamentals of CSS and introduce you to some coding constructs that will enable you to consistently produce well-organized, attractive documents. Though this is not intended to be an exhaustive treatment of CSS, we'll include instructions on how to code style sheets, how the cascade works and some practical examples. We also discuss the CSS box model.



This section of the guide has been extracted from our training material, and doesn't flow like the previous sections, which were originally written for our user guides. However, we have included the content because we felt it might be useful to some readers.

Why Use CSS?

CSS is a style sheet language that formats a document written in HTML. CSS files are designed to separate document content from document presentation, and are most commonly used to format elements such as color, font and layout.

CSS files are useful for:

1. Controlling the appearance of all elements and text on a page. You can use them instead of deprecated HTML tags such as <FONT>, <B> and <CENTER>.
2. Easily changing an entire page design through the use of external style sheet files.
3. Dynamically changing the appearance of a page. By using Javascript with event handlers, you can change the style object of any element on the page based on user responses.
4. Adding effects and design elements that are not possible with HTML tags such as:
   - Changing the appearance of your links.
   - Placing elements in exact positions on a page (either fixed or floating.)
   - Adding marquee effects such as shadows, drop-ins, fade-outs – the type of things you can do in PowerPoint.
   - Using layers to place objects on top of one another.
   - Adding custom scrollbars, table borders, etc.
5. Providing two looks for the same page: one for the display, one for the printer; without requiring you to code duplicate copies of the page.

Why You Should Use StyleSheets Now

Here are some of the reasons why you should be using style sheets now:

1. They are easy to write.
2. Future-proof' your code- some equivalent HTML tags are deprecated.
3. They are consistent to code. For instance, compare to older-style tags such as <CENTER> </CENTER> versus <td align=center>.

4. Coding with CSS is more. One style definition can affect the appearance of many elements on a page, and one stylesheet can affect the pages of an entire site.
5. They are great for debugging page layouts. For example, to show the borders on your tables, you can do that instantly without recompiling your programs or changing your HTML.
6. Pages that use CSS are more lightweight than those that use HTML for layout.
7. CSS allows better control over printing pages.
8. You can use it in conjunction with Javascript to add dynamic control over the behavior of a page.

Using Inline Style Attributes

While lacking some of the efficiency of external CSS files, you can also add style attributes inline using the `style=` keyword:

```
<table style="border : thin dotted #9370DB;">
```

Or:

```
<tr><td style="border : thin dotted #9370DB;">Row 1 Cell 1</td>
```

Using Inline Stylesheets

You can incorporate inline stylesheets using the <style></style> tag. You will generally place these between the <head> </head> tags.

Here is an example:

```
<html>
<head>
<style type="text/css">

 body   {background-color: white;
         margin-left: 0.5in;
         margin-right: 0.5in;}

</style>
</head>
<body>
<p>Page content goes here</p>
</body>
</html>
```

Using Links to External Stylesheets

An external stylesheet is a separate, standalone file, referenced by your page like any other file (the path is relative to the web server configuration).

The path to this file can be relative, and you will add the link between your <head> </head>.

For example, this page has a relative reference to the two CSS files:

```
<html>
  <head>
    <title>Maintain Customer Master File</title>
```

```
        <link rel="stylesheet" type="text/css" href="/websmart/v8.8/Idaho/css/screen.css"
media="screen, tv, projection" />
    <link rel="stylesheet" type="text/css" href="/websmart/v8.8/Idaho/css/print.css"
media="print" />
  </head>
<body>
   <div id="pagetitle" class="pagetitle"> Maintain Customer Master File</div>
```

While this reference is absolute:

```
<link type="text/css" href="http://www.cnn.com/css/style1.css">
```

As you can see there is lots of flexibility.

Stylesheet Media Types

A "media" attribute determines whether a stylesheet applies to a screen or a printed document, or a particular type of device.

This page has references to two separate CSS files:

```
<html>
  <head>
    <title>Maintain Customer Master File</title>
        <link rel="stylesheet" type="text/css" href="/websmart/v8.8/Idaho/css/screen.css"
media="screen, tv, projection" />
    <link rel="stylesheet" type="text/css" href="/websmart/v8.8/Idaho/css/print.css"
media="print" />
  </head>
```

When the user views the page on their monitor, the styles in file screen.css are used. When they print the same page using the print option from the browser, the styles in print.css are used.

When you design your site, you will typically use a smaller font size for printing than viewing, to fit more content on the printed page.

You can also do things like suppress printing images using the display:none; attribute in the print.css file.

For example, your print.css file could have this setting:

```
img {display: none; }
```

Using External StyleSheets to Change the Entire Page's Appearance

Consider this example:

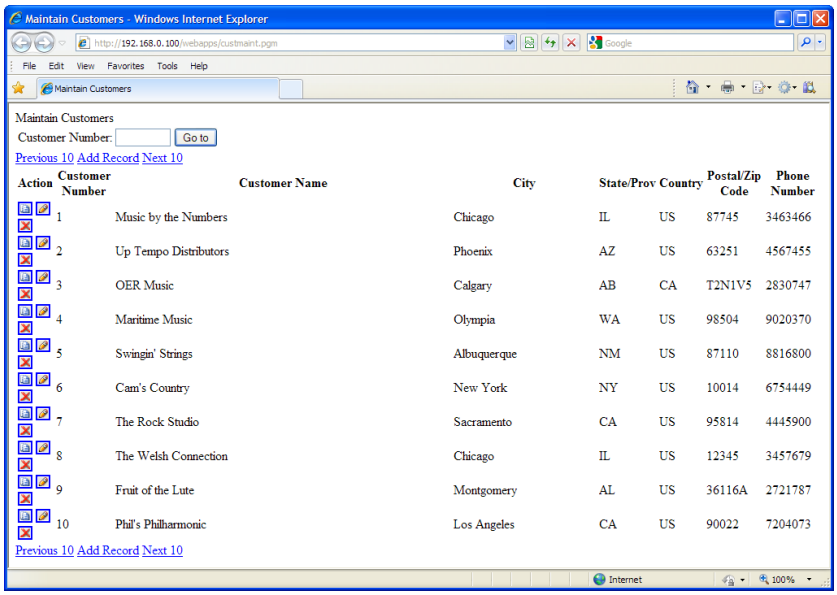


Figure 2-11. Page with no CSS applied.

Now consider this illustration, where a style sheet has been applied:

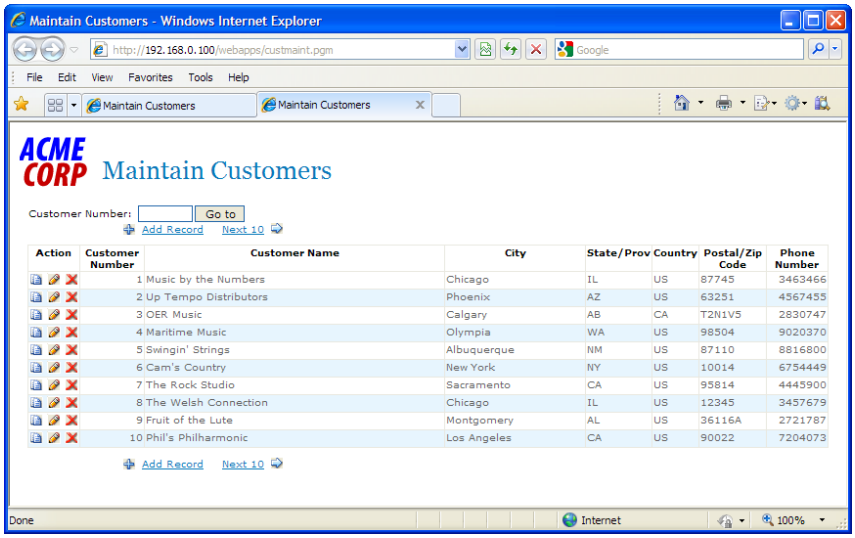


Figure 2-12. Same page with an external stylesheet called 'integrated' applied.

The effects added via the CSS file include:

- An image at top left of the page.
- A customized font style, size and color for the title.
- Customized submit buttons (Go to and Update).
- Images included with links for adding records or going to next 20 records.
- The links are centered on the page.
- There is a custom border around table.
- The table headings have been customized.
- Alternating row colors have been added to the table.
- The appearance of the input fields has been customized.

- The delete option is presented via a custom image, rendered as a link without the default blue border.

And here is the same page again, with a second CSS file applied:

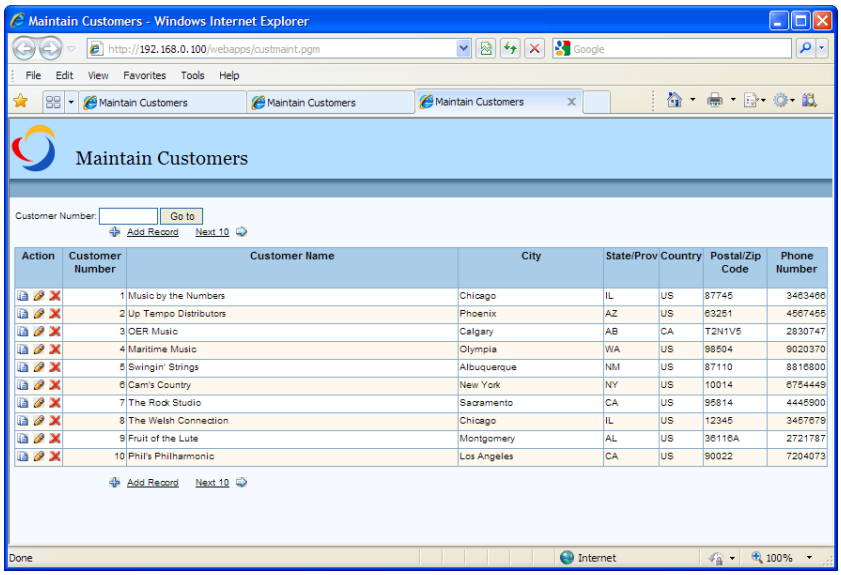


Figure 2-13. Same page with an external stylesheet called mennoblue applied.

In this illustration, there are some additional changes implemented via the CSS file:

- A customized background has been added to separate the heading from the page content.
- The table background also has been customized.

Stylesheet Coding Rules

The next sections describe some of the principal coding rules you should take into account when coding CSS.

Using Comments

Comments are written using /* and */. They can appear on a separate line from a CSS declaration, or at end of the same line.

Here are a couple of examples:

```
/* Change the indention of all paragraph tags */
p { margin-left: 10px; }

/* Change the background for a link when user hovers over it. */
a:hover {background-color: pink}
```

CSS Selectors

A 'selector' is a pattern used to match elements in an HTML or XHTML document to determine which style rules apply to elements in the document tree.

Common selectors include p, h1, table or div. These are examples of element type selectors that match any HTML element of type p (paragraph), h1 (heading 1), table (table) or div (div):

- **bold - Class selector.** This is an arbitrary name preceded by a period. You can associate this with specific element types, or any element type. You can referenced by more than one element in a document.
- **#maindiv - ID selector.** This is associated with a id attribute assigned to an HTML element. This should only be used once in a document.
- **div a - Descendant selectors.** For example, any link tag such as nested inside a div tag.
- **a:active, a:visited, a:hover.** These are dynamic pseudoclass selectors.

CSS specifications have the following general syntax:

1. Selector followed by declaration blocks (curly braces containing declarations). For example:

```
p {font-size: 11px; }
.indented { margin-left: 10px; }
```

2. CSS 'declarations' are coded inside curly braces { } with the attribute : value followed by semicolon ; such as these:

```
p { font : courier; font-size: 10px; }
p { font-size: smaller; }
```

Element Selectors

Elements are referenced by specifying the HTML element name, followed by the style sheet specification.

Background Image on a Page

Here is an example of how to add a background image to a page:

```
body {
        background-image : url(images/Picture1.jpg);}
```

Paragraph Settings

These are some sample paragraph settings:

```
p {
        font-family : Verdana, Geneva, Arial, Helvetica, sans-serif;
        font-size : 10px;
        margin-left: 0.5in;
        margin-right: 0.5in
        }
```

Table Settings

Here is an illustration of how to add a custom table border:

```
table {
        border : thin dotted #9370DB;
        background-color : #B0E0E6;
        }
```

Table Cell Settings

You could use this line to override your default CSS settings:

```
td {
        font-weight : bold;
        font-family : cursive;
        }
```

Link Settings

Use a line like this one to override your hyperlink colors:

```
a {color: green; }
```

You can use shorthand methods to code element selectors:

```
h1-h6, p, table, input, submit, textarea, select  {
        font-family: "Verdana, Arial, Helvetica";
}
```

This applies the font to any table, paragraph or h1, h2,h3,h4,h5 or h6 elements, and to any input form elements, submit buttons, text areas or drop down boxes.

Class Selectors

A class is a custom-defined style specification that can be referenced by any HTML element using the class= attribute.

Here is an example:

```
.indent {
```

```
        margin-left: 20px;
        background-color: #A0A7A4;
        font-style: italic;
        color: #8B008B;
}
```

Using this CSS, the following HTML:

```
<p>This is a normal paragraph. </p>
<p class="indent">Now is the time for all good men to come to the aid of the party</p>
<p>This is also a normal paragraph</p>
```

Is rendered as follows:

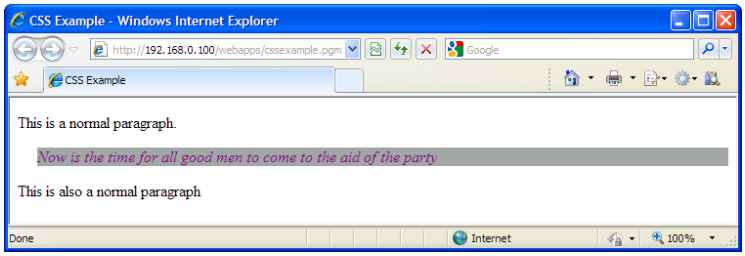


Figure 2-14. Example of a CSS tag applied to the middle line.

You can use a class with any element, or with a specific element. For instance:

```
.indent {margin-left: 5px; }
```

or you can also do this:

```
p.indent { margin-left: 10px }
span.indent {margin-left: 20px;}
```

ID Selectors

ID selectors are used with HTML elements that have a unique id attribute value. For example:

```
div#main { background-color: #f0f0f0; margin: 10px; }
```

or:

```
#main { float: left; background-image: ../images/logo.gif }
```



Since an id selector should always reference a unique id, the 'div' portion of the selector is really unnecessary – it's more for clarity in documenting usage.

Descendant Selectors

A 'descendant' selector is one that has an element, class or id selector that is a descendant of another element, class or id. Note that this is a more general type of selector than a 'child' selector. In other words, a 'child' is the first generation descendant of an element, while a 'descendant' can belong to any generation.



Child selectors are not supported in IE, and therefore are of limited use.

Here's how you code a child selector versus a descendant selector:

Child:

```
body > div { font-family: Arial; }
```

Descendant:

```
body div { font-family: Arial;}
```

Here is some sample HTML that incorporates this:

```
<body>
<div>This text would be in Arial font for both child and descendant selectors, while
<div>This text would be in Arial font for only the descendant selector</div>
</div>
</body>
```

Examples of More Complex Selectors

These examples are taken from a set of WebSmart templates that uses CSS extensively to control layout and appearance. Some selectors take advantage of the Cascade rules: that is, they have an accumulative effect. We've provided an explanation that follows each selector.

```
#contents {
        margin-left: 10px;
        margin-right: 10px;
}

/* For an element with an id of "contents", the left and right margin values are 10
pixels. Since the top and bottom margin values are not specified, these take on the
default values.  The element can be any HTML element, although it's intended for use
with a div, like this:
<div id="contents">Text sample. </div>

 */

#contents table { border-collapse: collapse; }
/* Any table element inside an element with an id of "contents" will have it's cell
borders collapsed (meaning the lines around cell borders appear as one, instead of
individual boxes per cell) – see images below: */

#contents td, #contents th { border: 1px solid #EAE9E1; }
/* td and th cells within tables in the "contents" element have a border on all four
sides that is one pixel wide, solid line in a custom color. */

#contents td.htext, #contents th {color: black; font-weight: bold; text-align: center;}

/* Within the "contents" div, any text inside a th cell, or a td cell with a class of
htext, is black, bold and centered.

#contents tr {
         color:#666666; vertical-align: top;
}
```

```
All cells in every row of a table inside the "contents" div have text that is a custom
color, and vertically aligned to the top of the cell. Because a td or th element has
higher ‘specificity’ in CSS rules, this rule does not override the previous one. */
#contents  tr.altcol1 {background-color: #ffffff;  }
#contents tr.altcol2 {background-color: #E7F5FE;}

/* All table rows with classes of altcol1 or altcol2 inside the "contents" div have
custom background colors as listed. */
```

Dynamic Pseudo-class Selectors

Here is an excerpt from w3.org, describing dynamic pseudo-class selectors:

"Pseudo-classes classify elements on characteristics other than their name, attributes or content; in principle characteristics that cannot be deduced from the document tree. Pseudo-classes may be dynamic, in the sense that an element may acquire or lose a pseudo-class while a user interacts with the document." [http://www.w3.org/TR/CSS21/selector.html#pseudo-elements]

The most common use of pseudo-classes in the current implementations of CSS is with links, in order to apply different styles to the different states of a link, such as link, active, hover or visited. The :link pseudo-class applies for links that have not yet been visited.

The :visited pseudo-class applies once the link has been visited by the user. Here is an example:

```
a:link, a:visited { text-decoration: none; }
/* remove underlines from links that have or have not been visited. */

a:hover {background-color: #dea63f; }
/* change the background color of a link when you hover over it. */
```

The CSS Box Model

The CSS Box model lets you control margins, paddings and borders according to the 'visual formatting model' defined by w3c standards.



See http://www.w3.org/TR/CSS21/visuren.html for further information. You can find another good overview at http://www.brainjar.com/css/positioning/.

The complete set of rules for the rendering of blocks is very complex so it's best to use trial and error to determine if you get the results you wish.

'Block' versus 'Inline' Elements

A block element is any HTML element that has, by default, an implicit carriage return at the end of it. In other words, any element following it is pushed to the start of the next line. Common block elements are:

- Paragraphs, headings (H1 – H6, for example), tables, divs and forms.
- 'Inline' elements cause content that follows them to continue to flow on the page layout, without inserting a carriage return.

The 'box model' works differently for 'block' versus 'inline' elements. Also, the width and height CSS properties generally only work reliably with 'block' elements.

The 'Cascade' in Cascading Style Sheets

Classes can also be associated with specific elements. This causes the element to inherit the style attributes of an element definition and override/add to it with the specific class attributes. For example, this CSS:

```
table {
        border : thin dotted #9370DB;
        background-color : #B0E0E6;
        }
td {
        font-weight : bold; font-family : cursive;
        }
table.customer { border: thin solid black}
td.customer {
        font : Verdana, Geneva, Arial, Helvetica, sans-serif;
        font-weight: normal; font-size: 10px;
        }
```

with the following HTML:

```
<table class="customer">
  <tr>
    <td class="customer">Row 1 Cell 1</td>
    <td>
```

```
      Row 1 Cell 2</td><td>Row 1 Cell 3
    </td>
  </tr>
  <tr>
    <td>Row 2 Cell 1</td>
    <td>Row 2 Cell 2</td>
    <td>Row 2 Cell 3</td>
  </tr>
</table>
```

is rendered as:

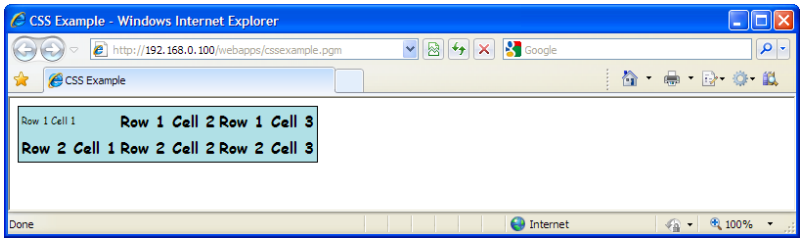


Figure 2-15. Combination of the CSS with some HTML.

You can see that the table inherits the class attributes of both the table element and the `table.customer` class, and the first cell (with `style="customer"`) inherits the td element and combines it with the attributes of the `td.customer` class.

In this case, we used a different font than the td element, so that overrides the earlier value. All other cells use the td element's style definition.

Elements can inherit some of the properties of their parent (container) elements. This is also part of the 'cascade' rules.



Note that the 'cascade' rules are arcane and may not be enforced consistently between browsers.

Here's a quick way to set the font for every element in a document, whether it's a table, paragraph, link, or some other element:

```
body { font-family: cursive; font-size: 11px;}
```

If we add this to the CSS in the last example, with this HTML:

```
<body>
<table caption="This is a Table">
  <tr>
    <td>Row 1 Cell 1</td>
    <td>Row 1 Cell 2</td>
    <td>Row 1 Cell 3</td>
  </tr>
  <tr>
    <td>Row 2 Cell 1</td>
    <td>Row 2 Cell 2</td>
    <td>Row 2 Cell 3</td>
  </tr>
```

```
</table>
<p>This is a normal paragraph. </p>
<p>Now is the time for all good men to come to the aid of the party</p>
<p>This is also a normal paragraph</p>
</body>
```

Is rendered:

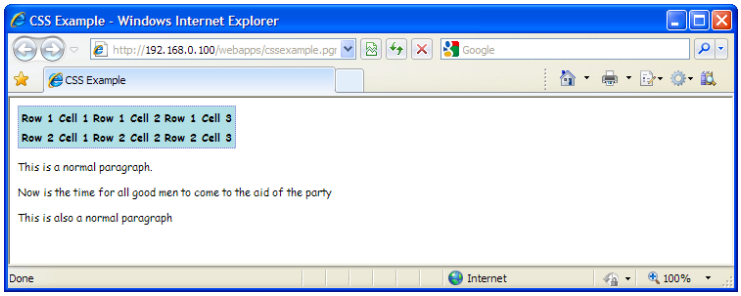


Figure 2-16. Cascade illustration.

Note that the size of the text in the <TD> tags is larger and bold. In this case the cascade only applies to the font-family.

If this seems inconsistent to you, it is!

The best way to try out the 'cascade' is to test the rendering by writing simple static pages to test your style sheets. Remember, once you've done this, you can use the same stylesheet in any page via an external link.

Saving Style Sheet Specifications to External Files

So save your CSS file to an external file and re-use it in various pages across your site, simply place your style sheet in a text file (using Notepad or some other editor) and save it to your IFS (or another web server) in the desired directory with a .css extension.



You don't need the <style> </style> tags, or the <!-- --> comment tags.

CSS Best Practices

Below are some suggested CSS best practices.

Use TopStyle Lite to help you write your style sheets.

We have found TopStyle Lite (www.bradsoft.com) to be a great CSS editor.

Use external style sheets for common selectors.

Re-use attributes as much as possible, by sharing the CSS file across your site.

The statement goes in <head> </head> section, like this:

```
<head>
        <title>ZEN24ICPAAT Maintenance</title>
        <link rel="stylesheet" type="text/css" href="/websmart/v8.8/lincoln/css/screen.css" />
</head>
```

Use the media attribute to control different styles for screen versus print.

This allows you print with different attributes than you use on the screen.

```
<head>
        <title>ZEN24ICPAAT Maintenance</title>
        <link rel="stylesheet" type="text/css" href="/websmart/v8.8/lincoln/css/screen.css"
media="screen, tv, projection" />
        <link rel="stylesheet" type="text/css" href="/websmart/v8.8/lincoln/css/print.css"
media="print" />
</head>
```

Use inline styles for customized page features.

When you want to override your site-wide style sheet, you can also do this in your header:

```
<head>
  <meta name="generator" content="WebSmart" />
  <meta http-equiv="Pragma" content="no-cache" />
  <title>ZEN24ICPAAT Maintenance</title>

  <link rel="stylesheet" type="text/css" href="/websmart/v8.8/lincoln/css/screen.css"/>

  <style type="text/css">
        <!--
        H1      {font: 20pt Arial;
                 color: black;
                 text-align: center;
                 font-weight: bold}
        -->
  </style>
```

In this illustration we have set a special look for the H1 heading.

Override standard HTML selectors where possible.

When you need to override standard selectors, approach this way:

```
h1 {font-size: 12px; color: Fuschia;}
<h1>This is Fuschia</h1>
```

instead of:

```
.myhdg1 {font-size: 12px; color: Fuschia;}
<div class="myhdg1">This is Fuschia</div>
```

or you can also do this:

```
h1.myhdg1 {font-size: 12px; color: Fuschia;}
<h1 class="myhdg1">This is Fuschia</h1>
```

Avoid having to specify inline styles by using descendant selectors.

```
div#portlet a {text-decoration: none; }
```

HTML:

```
<div id=“portlet”>
        <a href=“menu1.htm”>Menu 1</a>
        <a href=“menu2.htm”>Menu 2</a>
```

```
        <a href="menu3.htm">Menu 3</a>
</div>
```

Avoid using inline styles as attributes of elements.

Use this approach:

```
.custom1 { font-weight: bold; font-size: 13px; }
<p class="custom1">This is bold, 13 px size</p>
```

instead of:

```
<p style="font-weight: bold; font-size: 13px;"> This is bold, 13 px size</p>
```

Use lower case names for custom selectors.

Use this:

```
.mydiv1
```

rather than:

```
.MyDiv1
```

or:

```
.Mydiv1
```

Use relatively short names for custom selectors.

It's better to use a name like:

```
.mydiv1
```

than:

```
.mymaincontainerdivb4table
```

CSS Magic

Using CSS you can create some cool effects on your pages that are not possible with HTML `<font>` tags. These include:

- Faking an input text box
- Simulated buttons
- Changing the look of forms controls
- Cool hyperlinks
- Making sure forms don't occupy space on the page
- Layering fragments of pages on top of one another
- Dynamically changing the appearance of a page
- Faking the appearance of output text to look the same as an input text box

Below are some examples.

Making output-only data appear the same as input-capable data.

If you want output-only data to appear the same as input-capable data you can define a style sheet as follow:

```
.fakeinput {
```

```
        border-top : 2px;
        border-left: 2px;
        border-bottom : 1px Silver;
        border-right: 1px Silver;
        border-style : inset solid inset;
        font-size : 10pt;
        font-family : "MS Sans Serif", Geneva, sans-serif;
        padding-top : 1px;
        padding-bottom : 1px;
        padding-right: 3px;
        padding-left: 1px;
}
```

Compare the following two input boxes:

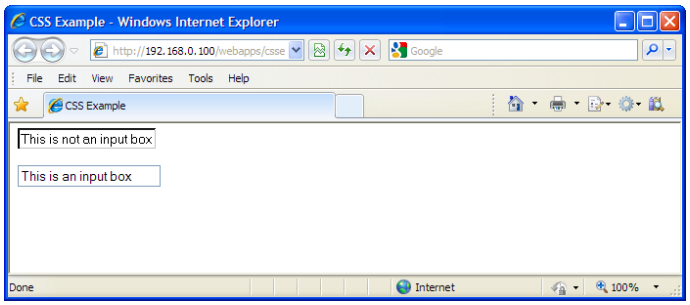


Figure 2-17. Illustration of an input box.

The first is not an input box at all. We used this code:

```
<span class="fakeinput">This is not an input box</span>
```

The second used this code:

```
<form>
<input type=text name="inputbox" value="This is an input box">
</form>
```

This is useful for making output-only fields look the same as input-capable fields. It also shows you how you can combine several stylesheet properties to affect the appearance of an element.

Note also that we used the span tag, which allows us to affect inline elements (those that do not have a beginning and ending tag). In this case, the span just included straight text.

How To Create Pseudo-Submit Buttons

Sometimes you need more flexibility in submitting forms than is available using standard submit buttons. Or, to avoid having any form control at all, you can use a link. A standard hyperlink in html is defined as follows:

```
<a href="mypgm.pgm?task=update&custno=0001">Update</a>
```

or:

```
<a href="http://www.progenwebsmart.com">Come See Our Site!</a>
```

The first 3 examples below use anchor tags. Example 4 is a true submit button, just to demonstrate how it is possible to make both look almost identical. The screen-shot below shows all four examples.

Example 1: Using a class for a surrounding table cell element, and another for text.

You can add this CSS to your page:

```
.bttn {
        border-style:outset;
        border-color:#9999cc;
        border-width:2px;
        background-color:#cccccc;
        text-align:center;
        width : 90px;
        height: 10px
}
```

And this one for the text inside an anchor <a> tag:

```
.snu {
        font-family:arial,san-serif;
        font-size:11px;
        text-decoration:none;
        color:#000066
}
```

Here's the HTML:

```
<table>
  <tr>
        <td class="bttn" noWrap>
   <a href="http://www.bcdsoftware.com/nhbchkout.pgm?TASK=Proceed">
   <span class="snu"> Proceed </span></a>
   </td>
  </tr>
</table>
```

Example 2: A a single style for the button design and text contents.

The class is specified in the <td> tag. Notice that the text inside the button look like a link. Even though we've used a style with text-decoration: none, this doesn't take effect because the anchor (<a>) tag is at a different level in the CSS hierarchy.

The stylesheet specification is:

```
.sbttn {
        border-style:outset;
        border-color:#9999cc;
        border-width:2px;
        background-color:#cccccc;
        text-align:center;
        width : 60px;
        font-family:arial,san-serif;
        font-size: 10px;
        text-decoration:none;
        color:#000066
}
```

The HTML code is:

```
<table width="90%">
  <tr>
   <td class="sbttn" noWrap>
   <a href="http://www.excelsystems.com/nhbchkout.pgm?TASK=Proceed">
     Proceed </a>
    </td>
    <td> </td>
    <td class="sbttn" noWrap>
      <A href="http://www.excelsystems.com/nhbchkout.pgm?TASK=Exit">
        Exit </a>
    </td>
    <td> </td>
  </tr>
</table>
```

This is illustrated below.

Example 3: No table – just an anchor tag.

This is the most elegant implementation of the button. It just uses an anchor (<a>) tag with the class specified inside. The style has a fixed width of 60 pixels.

The stylesheet specification is:

```
A.sbttn {
        border-style:outset;
        border-color:#9999cc;
        border-width:2px;
        background-color:#cccccc;
        text-align:center;
        width : 60px;
        font-family:arial,san-serif;
        font-size: 10px;
        text-decoration:none;
        color:#000066;
}
```

The HTML is simply:

```
<a class="sbttn"
href="http://www.excelsystems.com/nhbchkout.pgm?TASK=Proceed">Proceed</a>
```

This is also illustrated below.

Example 4: Same as the anchor, but using a Submit button.

The stylesheet specification is identical to that for the anchor:

```
.sbttn {
        border-style:outset;
        border-color:#9999cc;
        border-width:2px;
        background-color:#cccccc;
        text-align:center;
        width : 60px;
        font-family:arial,san-serif;
        font-size: 10px;
        text-decoration:none;
        color:#000066
}
```

The HTML is:

```
<form action=”pgm.pgm”>
<input  class=”sbttn” type=submit value=Proceed name=task>
</form>
```

Here’s the results of all four examples:

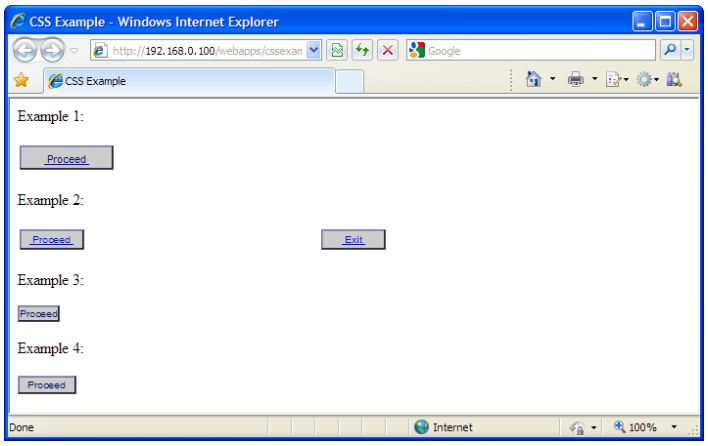


Figure 2-18. Results of the 4 examples.

Changing the Appearance of Forms Controls

You can also use CSS to control the appearance of the borders around input text boxes, drop down boxes, etc.

Example: Input text box with simple blue border, no bevel.

The style sheet specification for this is:

```
INPUT.box {
        font-family:Verdana, Arial, Helvetica;
        font-size: 8pt;
        border:1px solid blue;
}
```

The HTML is:

```
<form>
<input class="box" type="text" value="Input goes here">
</form>
```

The resulting page looks like this:

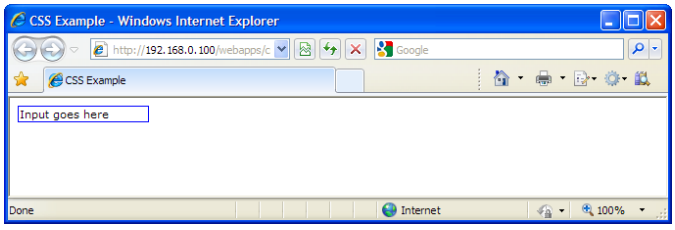


Figure 2-19. Input text box with simple blue border.

Hyperlinks

We've already seen some examples of how you can change the appearance of hyperlinks with the pseudo-buttons. One of the most frequently used features is text-decoration: none; which eliminates the underline on the links.

Another concept with style sheets is the pseudo-class, such as:

```
a:hover
```

This indicates that the class takes effect when the user hovers the mouse over the link.

Other possible values include:

- a:visited – User has visited the link (usually, without a stylesheet, these links appear as red underlined.
- a:link - unclicked state, no mouse hovering.

The <DIV> and <SPAN> Tags

The <DIV> and <SPAN> tags perform similar functions. They are used to surround an element or collection of elements on the page in order to provide page layout control, including positioning and appearance on the page.

The <DIV> tag can surround any element or set of elements, while the <SPAN> tag can only surround and influence the appearance of text. Both are most useful when used in conjunction with CSS.

A good example of using the <DIV> tag is to conditionally hide or reveal a portion of a page by dynamically setting its style settings.

The tag should be used for formatting portions of text only, and should be used in place of several deprecated HTML tags, such as:

- <FONT>
- <CENTER>
- <B>
- <I>
- <BIG>
- <STRONG>

Neither of these tags actually occupies any screen real-estate. They are simply used for organizational purposes.

Here's an example using a <DIV> to place an opaque layer on a specific portion of the page at a specific location:

```
<div style="position:absolute; left:50px; top:75px; width:200px; background-
color:gray;"><p> The background is gray, and the box produced by the div is at left of
50, top of 75 pixels relative to the document window.</div>
```

Like so:

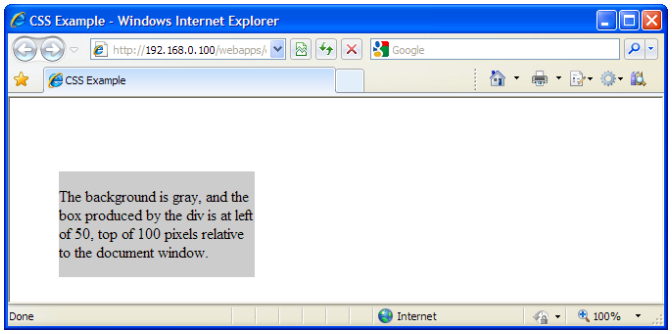


Figure 2-20. Manually positioning a custom box on the page.

Appendix L - Source Code Copyrights and Attributions

This appendix includes all source code attributions.

Scintilla and SciTE

License for Scintilla and SciTE

Copyright 1998-2002 by Neil Hodgson <neilh@scintilla.org>

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

NEIL HODGSON DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL NEIL HODGSON BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Firebug

Software License Agreement (BSD License)

Copyright (c) 2007, Parakey Inc.

All rights reserved.

Redistribution and use of this software in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Parakey Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission of Parakey Inc.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Blowfish

Blowfish implementation copyright 1997 by Paul Kocher, and used under LGPL 2.1.

Newtonsoft.Json

The MIT License (MIT)

Copyright (c) 2007 James Newton-King

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CRCpp

CRC++

Copyright (c) 2016, Daniel Bahr

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of CRC++ nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Apache License

Apache License Version 2.0,

January 2004

http://www.apache.org/licenses/

1. Definitions. "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document. "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License. "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity. "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License. "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files. "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types. "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an

example is provided in the Appendix below). "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof. "Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution." "Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions: (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and (b) You must cause any modified files to carry prominent notices stating that You changed the files; and (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may

add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

Index

.

.NET Framework 14

1

10061
  Exodus51 error code 17

5

5250 Emulation options 45

A

A
  element type 240
Adding New Page Elements
  using visual editor 172
Admin Functions
  URL to run 38
Aligning Elements 161
Anchor
  element type 240
APACHE
  HTTP configuration 401
  XL_PRESTO server instance 11
Auto Detection Settings
  assessing effectiveness (tutorial) 114

Auto Formatting
environment settings 43
screen customization option 139
Auto Logins
creating and working with 88
how to use 90
library list requirements 91
security considerations 92
template included with Presto 92
Auto Panel Detection Settings
illustrated explanation (tutorial) 116
Auto Screen Detection
80 and 132 col display settings 41
Autocompletes 201
custom HTML, implementing via 301
SQL examples 205

B

Background Color
setting in HTML 499
BeforeLoadGrid
overriding jQGrid properties 341
Blanks
inserting in the HTML 493
Block Container (div)
element type 210

Bold
making text so in HTML 497
Button
adding to custom HTML 286
attaching events 214
element type 214
function key type 47
Buttons
creating with menulinks transformations 438

C

Cascading Style Sheets
See CSS 512
CCSID
for virtual terminals 20
setting 20
CCSID Settings
virtual terminals 398
Change Management 379
copying changes between environments 384
Character Attributes
controlling in the HTML with style sheets 512
Checkbox
description 250
Class Attribute
for custom CSS 194
HTML span tag (tutorial) 317

clickableoptions
transformation options 433
Colors
background colors 499
named colors & RGB values 499
using in pages (HTML ref) 499
visual editor property transformations 187
column
HTML span tag (tutorial) 317
Column Widths
controlling in tables (HTML ref) 505
Columns
adding to custom tables 146
Command Lines
in custom HTML (tutorial) 318
Concatenating
using the append function 448-450, 454-455
Conditional Fields
impact on customizations (tutorial) 118
config.bws
configuration file, location 398
Confirming your IBM i Installation 13
Connection
testing Presto Designer connection 38
Connection Name
environment 38

Copy Attributes
setting to false 199
Copying Changes Between Environments 384
Copying Transformations to Another Page 164
CSS
concept overview 512
referencing with class attributes 194
transformation property 432
CSV
exporting data in csv format 220
Current Library
for auto logins 89
Custom HTML
adding command keys 286
adding drop down boxes 288
attaching 75
autocompletes, implementing manually 301
command lines (tutorial) 318
creating 314
error message subfile 318
function keys in the HTML 317
HTML coding basics 316
JavaScript 281
managing entries 74
optimizing initial code 325
option fields 318

screen values, using 299
SQL queries 80
Custom Login Applications 91
Custom Tables
setting perimeter 146
where Presto builds them 144
Customizations
attached incorrectly or missing (tutorial) 114
exporting and importing transformations 164
Customizing Pages
discussion of options 33

D

Data Scrubbing 46
Data Source
export data transformation 220
Date Picker
element type and example 223
datepicker
localization settings 20
Debug Logs
capturing 406
Design
HTML development tools 490
IBM i vrs. web pages 489
Disable Output Only Screens
setting 44

Div
element type 210
Document Object Model
See DOM 511
DOM 511
Dots
replacing leading dots with colons 43
Downloading Presto 7
Drop Down Box
element type & options 226
Drop Down Boxes
adding to custom HTML 288
dynamic drop down box example 229
SQL queries, using with 230
static & dynamic values, combining 232
static drop down box example 227-228
Dynamic Text Values
adding to the HTML 329

E

Elements
adding with visual editor 172
aligning together 161
moving 160
properties 163
removing 173
resizing 169

resizing input fields (example) 169
selecting in the visual editor 159
setting the type 166
EnableStrPcCmd 99
Entry URL
for auto logins 90
Environment
configuring 37
connection name 38
library 38
port number 38
screen detection settings 41
server address 38
server connection settings 38
testing connections 38
URL 38
Environment Settings
function key type 47
window creation 43
Error Message Subfile
on custom HTML 318
events
transformation property 434
Events
attaching to buttons 214
example adding to a select box 289

visual editor property transformations 189
Executing PC Applications 99
Exit Points
for auto logins 90
Exit URL
for auto logins 89
Exodus51
communications errors 17
starting the subsystem 12
starting the subsystem after an IPL 12
Export Data
data format options 220
data source property 220
description & examples 264
Exporting Transformations 164

F

Favicon 96
Field IDs
usage (tutorial) 317
Field Settings
editing element properties 191
Fields
resizing 169

Files
selecting in visual editor
Visual Editor
browsing for files 188
Find Window 65
Flyout Menu
customization options 411
Fonts
changing sizes in HTML 498
controlling in the HTML with style sheets 512
formatting in HTML 497
format
menulinks option 438
Forms
HTML reference 507
text input fields ref 508
Function Key Type 137
environment settings 47
Function Keys
in custom HTML 317
showing the number in buttons 48
Function Keys in Protected Fields
settings 48

G

Global Identities 124
Group Selection 174

H

Handler Program

RPG OA 337

Heading Tags in HTML 495

hide_target

transformation property 437

hide_value

transformation property 436

HTML

adding dynamic values to 329

adding images to pages 498

background colors 499

basic guide to editing custom HTML 316

body tags 494

bold tag 497

creating and storing files 491

custom Presto tags 320

development tools 490

exporting data in HTML format 220

font formatting 497

font size adjustments 498

formatting monospaced data 497

head tag 494

heading tags 495

indenting the code 493

inserting blanks 493

italicizing text 497
overview of standard Presto-generated code (tutorial) 316
paragraph tags 497
pre-formatted text tags 497
reference guide for green screeners 489
strong tag 497
tables reference 502
testing and troubleshooting 491
text input fields ref 508
title tag 494
transformation properties 431
using color in your pages 499
using links 500
HTTP Server Configuration
Powered by Apache 401

I

Icon
element type (visual editor) 239
id
HTML span tag (tutorial) 317
Identification Properties 195
iframe
element type 240
IFS
folders installed 7

Images
adding image elements 240
adding to pages (HTML ref) 498
Importing Transformations
from another page 164
Include Fields in Screen Identifier 42
Indention
of HTML code 493
Inline Container (span)
element type 242
Insights 456
Installation
confirming 13
overview 7
transferring the server install library 7
Interactive Pages
using forms and forms controls (HTML reference) 507
Italicize
text in HTML 497

J

JavaScript
adding to custom HTML 281
concept overview 511
jqGrid
OA programs 341
jQuery 281

JSON string 186
building with external program 304

K

Kerberos
implementing 88

L

Label
element type 243
Language Support
adding multi-language support to Presto 399
Launching Current Site 64
Leading Dots
replacing with colons 43
Library List
auto login requirements 91
License Key
entering with UNLOCK command 7
License Key Server 464
Limit Capabilities
user profile option, and auto logins 88
Links
function key type 47
navigating using (HTML ref) 500
Localization
datepicker 20

Login Applications
coding custom programs 91
Login Options
Presto and Nexus Portal 88
Logo
changing in the skin (tutorial) 350
Lost Customizations
understanding the cause (tutorial) 114

M

Macros 84
attaching to screen elements 85
commands 87
managing and editing 86
recording 84
replaying 85
saving or discarding 84
Manage Pages 73
attaching custom HTML 75
Mark All
Search option 65
MDCMS 469
Menu
for autologins 89
Menu Links
detecting and building 138

menulinks
creating buttons 438
Microsoft .NET Framework 14
minLength
autocomplete parameter 202
Missing Customizations
understanding the cause (tutorial) 114
Monospaced Data
formatting in HTML 497
Moving Changes from one Environment to Another 384
Moving Elements 160
Multi-Language Support 399

N

Named Colors 499
Nexus Portal
building auto logins for 88
using as your auto login handler 90
No Message Subfile
screen customization option 141

O

OA 335
considerations 336
handler program F spec 337
jqGrid 341
Presto OA Grid API 342

RPG program requirements 337
Option Fields
in custom HTML 318
options
transformation property 432
Output Only Screens
customizing 44

P

Padding Properties 196
Page Description
editing and saving 63
Page Identities
creating (tutorial) 122
overview 68
Page Tab
reference 67
Palette
adding custom entries 180
using to add new page elements 179
paneldescriptionend
advanced configuration option 55
paneldescriptionrow
advanced configuration option 54
paneldescriptionstart
advanced configuration option 54

Password Field
element type 226
Passwords
HTML form input types 508
pattern_label
transformation property 439-440
pattern_label_group
transformation property 437
pattern_value
transformation property 437
pattern_value_group
transformation property 437
PC Applications
executing from Presto 99
PC Components
installing 14
Port
environment connection 38
for Presto Designer connections 16
Position Properties 197
PR_CONVQ
data queue issues 404
PR_HUB 407
prefix
transformation property 429

Presto
Apache HTTP server instance 7
Apache HTTP server instance (starting) 11
components 7
Downloading 7
IBM i libraries 7
IFS folders 7
installing IBM i components 8
runtime architecture 4
Presto Designer
adjusting the layout 66
configuring IBM i connections 15
launching 36
reference section 33
restoring hidden sections 66
Presto IBM i Jobs 407
Presto Session ID
validating 409
Presto Skins 331
Presto.CloseSession 446
presto.js 421
Presto.Oa.Grid.GetField 342
Presto.OpenSession 446
Presto.Transform
function reference 421
Previewing SQL Query Results 79

Program
for auto logins 89
Properties
expanding & collapsing all 163
reference section 163

Q

QQF0* Jobs 407
QTMHHTP1 407
QTMHHTTP Message Queue
Presto authority requirements 14
QUSRWRK 404
QUSRWRK Subsystem
system requirements 14

R

Radio
description & example 240
dynamic options, adding 248
static set of options, adding 247
Radio Buttons
aligning vertically 248
Regular Expression
in Find window 66
Renaming a Presto Library 396
Resizing
elements 169

Retain Spaces in Fields option 45
RGB
values for named colors 499
RNMPRLIB
renaming a Presto library 396
row
HTML span tag (tutorial) 317
RPG OA 335
Runtime Architecture 4

S

Screen Design
IBM i vrs. web pages 489
Screen Tab
reference 67
Screen Values
custom HTML, using in 299
using in visual editor 176
Screen, Field and Attr Tags 320
Scrub Data
environment option 46
Search Options 65
Security Considerations
when using auto logins 92
Select Boxes
events, adding 289
Selecting Multiple Items 174

Selector Properties 423
SendRequest
HTML generated by Presto (tutorial) 318
Separator
subfile option separator config 411
Server Address 38
Server Connection Settings
reference section 38
Server Settings
timeouts 39
WebFacing port 40
Server Side Data Validation 45
Show Function Key Number 48
page setting 138
screen customization option 137
Size of Fonts
adjusting in HTML 498
Skins
logo, changing (tutorial) 350
switching your environment skin 49
SmartCharts
combining a static structure with dynamic values 298
dynamic charts with SQL 296
element type 250
static charts 297

Spaces
inserting blanks in the HTML 493
Span
element type 242
Spool File Listings 93
SQL
autocomplete examples 205
creating and working with queries 77
custom HTML with queries 80
drop down boxes 229-230
export data feature 221
generating dynamic SmartCharts 296
parameters, using with autocompletes 208
preview 79
src
transformation property 432
SSO
Single Sign-On 88
Storing HTML Files 491
STRPCCMD 99
STRPCO 99
Style Sheets
concept overview 512
Subfile Flyouts
customizing 411

Subfile Options
detecting and building 141
setting the separator 411
Submit on Enter
text areas 263
suffix
transformation property 429
System Requirements 2

T

Tab Index
setting 191
Tables
controlling table widths (HTML ref) 505
creating and working with custom tables 143
editing with visual editor 175
HTML reference 502
where Presto builds custom tables 144
Tabs
breaking page content into 257
page tab 67
target
transformation property 436, 439
Text Area
element type 250
submit on Enter 263

Text Fields
element type 264
Text Fonts
formatting in HTML 497
Text Input Fields
HTML ref 508
Text Properties 198
text_pos
transformation property 433
Timeout
browser 39
server settings 39
to_object
transformation property 425
Transform Backslash Expressions 66
Transformations
expanding & collapsing the list 163
Exporting and importing 164
function reference 421
resetting 164

U

Uninstalling a Presto Library 396
UNLOCK
license key entry 7
URL
for running the Presto Designer admin pages 38

Use WebFacing Connection 39
userresources
customization directory 281

V

validateusersession
reference 409
Validation
serve side 45
value
object property 429
Variables
getting and setting Presto environment variables 442
vertical-inputs
aligning radio buttons vertically 248
Virtual Terminals
CCSID settings 398
CCSIDs 20
installation requirement 14
Visual Editor
adding new elements 172
aligning elements 161
element properties reference 163
events, adding 189
moving elements 160
reference section 157, 344
removing elements 173

resizing elements 169
right click option to add elements 172
screen values, using 176
tables, editing 175
using 157, 344

W

WebFacing
configuring 397
requirements 397
troubleshooting 397
WebFacing Errors
troubleshooting 407
WebFacing Port
server settings 40
WebSphere
required components 397
Window Creation
environment settings 43
Windows
custom windows 147
restoring lost windows in Presto 66
WRKACTJOB JOB(QQF0*) 407

X

XL_PRESTO
default environment and product library 38

HTTP server instance 11