

QAD CSS (Customer Self Service): Developer's Course

TRAINING GUIDE



70-2910A
MFG/PRO eB2
September 2005

This document contains proprietary information that is protected by copyright. No part of this document may be photocopied, reproduced, or translated without the prior written consent of QAD Inc. The information contained in this document is subject to change without notice.

QAD Inc. provides this material as is and makes no warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. QAD Inc. shall not be liable for errors contained herein or for incidental or consequential damages (including lost profits) in connection with the furnishing, performance, or use of this material whether based on warranty, contract, or other legal theory.

Some states do not allow the exclusion of implied warranties or the limitation or exclusion of liability for incidental or consequential damages, so the above limitations and exclusion may not be applicable.

PROGRESS® is a registered trademark of Progress Software Corporation. Windows™ is a trademark of Microsoft Corporation.

MFG/PRO® is a registered trademark of QAD Inc. QAD, QAD eQ, and the QAD logo are trademarks of QAD Inc.

All other products and company names are used for identification purposes only, and may be trademarks of their respective owners.

© Copyright 2005 by QAD Inc. All Rights Reserved.

70-2910A

QAD Inc.

6450 Via Real
Carpinteria, California 93013
Phone (805) 684-6614
Fax (805) 684-1890

elearning@qad.com
<http://elearningportal.qad.com>

Contents

CHAPTER 1 INTRODUCTION	13
What Is CSS?	14
Introduction	14
Product Demonstration	15
Course Overview	16
CSS Layers	17
CSS Installation	18
CSS Implementation	19
CSS Architecture	20
Internationalization	24
Case Study Overview	25
 CHAPTER 2 INSTALLATION	 27
Installation	28
Installation Guide	28
Pre-Trip Questionnaire	29
Installation Planning Sheet	30
System Requirements	31
Hardware and Networking	32
Java and Disk Space	33
Client/Server Compatibility	34
Licensing Information	35
CD-ROM Contents	36
CSS Directory Structure	37

Product Installation Demo	38
CHAPTER 3 IMPLEMENTATION.....	39
Implementation	40
Pre-Planning	40
Security Group Maintenance	41
User Maintenance	42
Setting Up Email Events	43
CSS Registry	44
What Is It?	44
Where Is It Used?	44
How Does It Work?	45
Common Flags	47
Lab Exercise	48
Item and Customer Replications	49
CHAPTER 4 CSS LAYERS	51
CSS Layers	52
Overview	52
UI Layer	53
UI Business Rules Layer	54
EX Business Rules Layer	55
EX Data Layer	56
Adapter Layer	57
Layers Interaction	58
Customization Layer	61
Modification Layer	62
CHAPTER 5 TECHNOLOGIES APPLIED	63
Introduction	64
Progress 4GL	64
Web Technologies	65
Preprocessors	66
Global vs Scoped	67

Preprocessor Example	69
Conditional Compilation	70
Preprocessors in CSS	71
Code Example	72
Temp-Tables	74
Persistence	76
Persistent Option	78
Procedure Handles	79
Persistent Attributes	80
Code Demonstration	81
Super Procedures	83
Introduction	83
Called Procedure Example	84
ADD-SUPER-PROCEDURE()	85
Super Procedure Example	86
Business Logic	88
ADD-SUPER-PROCEDURE Method	90
UDF Declaration	91
UDF Usage	92
UDF Code	93
Procedure Calls	94
RUN SUPER Statement	96
Stacked Super Procedures	97
Business Objects	98
NO-ERROR Option	99
More Business Objects	100
Dynamic Objects	102
Introduction	102
Attributes and Methods	103
Query Handles	104
QUERY-PREPARE() Method	105
QUERY-OPEN() Method	105
Query Settings	106
Dynamic Queries and Buffers	107
CREATE Widget Statement	107

Buffer Handle	108
SET-BUFFERS () Method	108
Dynamic Query Manipulation	109
GET-FIRST(), GET-LAST(), GET-PREV(), GET-NEXT() Methods	109
Dynamic Buffer Fields	110
BUFFER-FIELD() Method	110
NAME Attribute & STRING-VALUE() Method	111
Publish and Subscribe	112
Introduction	112
Benefits	113
Named Events	114
Dynamic HTML	115
Lab Exercises	117
 CHAPTER 6 CSS ARCHITECTURE	 119
Naming Conventions	120
Name Format	120
Variable Name Codes	121
Internal Procedures and Functions	122
Comments	123
Lab Exercise	124
Data Architecture	125
CSS Superstructure	126
CSS Startup	130
 CHAPTER 7 CUSTOMIZATION	 135
Customization	136
Overview	136
Basic Template	137
Components	138
lib/lib_std.i	139
{tt_tablename.i}	141
putEXHeader	142
insertNewHeader	143

Panel Generation	144
insertNewFooter	145
Create New Program	146
Case Study, Lab 1	147
Page Navigation and Posting	148
Case Study, Lab 2	149
Maintenance Template	150
Overview	150
Additional Features	151
deleteRecord	152
getParentJoin	153
genLeftMaintenance	154
genCenterMaintenance	155
genRightMaintenance	156
New Features in Edit Template	157
putEditHeader	158
putEditButtons	159
setLocalRecord	160
genEditCenter	161
Create New Maintenance Program	162
Case Study, Lab 3	163
Report Template	164
Overview	164
Additional Features in rpe Template	165
genCenterReport	166
createInitParameters	167
Additional Features in rpt Template	168
Create New Report Program	169
Case Study, Lab 4	170
Core Objects	171
Common Features	171
Browse Object	172
Calendar Object	173
Event Object	174
Session Object	175

QAD CSS (CUSTOMER SELF SERVICE): DEVELOPER'S COURSE

Registry Object	176
ProcessError Object	177
Case Study, Lab 5	178
Move to Supers	179
Overview	179
Creating New Layer	180
Move Custom Procedures	181
Case Study, Lab 6	182
Customizing Your Procedures	183
Case Study, Lab 7	184
Modifying Core Product	185
Case Study, Lab 8	185
APPENDIX A LAB ACTIVITIES	187
Chapter 3 Labs	188
CSS Registry Lab	188
CSS Registry Lab Answers	190
Chapter 5 Labs	191
Lab-5a	191
Lab-5b	192
Lab-5c	193
Lab-5d	194
Lab-5e	194
Chapter 6 Labs	195
Naming Conventions	195
Chapter 7 Labs	196
Set-up Instructions	196
Case Study - Lab 1	197
Case Study - Lab 2	200
Case Study - Lab 3	202
Case Study - Lab 4	205
Case Study - Lab 5	207
Case Study - Lab 6	213
Case Study - Lab 7	213

Case Study – Lab 8	213
APPENDIX B ADDITIONAL CSS APIs	215
dataEngine	216
genNavigation	217
getDatalate	217
getKeyValue()	218
getPhraselate()	218
getPUK	218
getValue() (UI layer)	219
getValue (EX layer)	219
getXlate()	219
processEvent	219
putBrowse()	220
putCalendar()	220
putCheckBox	221
putInputText	222
putTextArea	223
setKeyValue()	223
setProcessError()	224
Examples:	225
APPENDIX C INTERNATIONALIZATION	227
Purpose	228
Translation	228
Database Tables	228
Images	228
Translation	229
Html Page - Header set up	229
Character Set	229
Number format	229
Date format	229
Html Page - Labels Translation	229
Javascript Messages Translation	230

QAD CSS (CUSTOMER SELF SERVICE): DEVELOPER'S COURSE

Implementation	230
Progress Message Translation	230
Implementation	230
Database Translation	231
Add Language Code to Index	231
Data Translation	231
Wpro_registry	231
Images	232
Implementation	232
Functions Used In Internationalization	233
getDatalate()	233
putCharSet()	234
getPhraselate()	235
getXlate()	236
getPUK()	238
setProcessError()	239
getDataValue()	241
APPENDIX D INTERNATIONAL INSTALLATION.....	243
Requirement	244
Installation Procedures for Business Partners	245
APPENDIX E CSS TABLES	249
APPENDIX F DEBUG MODULE	255
Overview	256
Stack Trace	256
Stack Trace Features	256
General / Specific Messages	257
{&EXDEBUG}	257
setExDebug	257
APPENDIX G STANDARDS	259
File Names	260

Temp-Table Standards 261

Defined Variables 261

Internal Procedures & Functions 262

Comment Body 263

{&EXDEBUG} Preprocessor 266

QAD CSS (CUSTOMER SELF SERVICE): DEVELOPER'S COURSE

CHAPTER 1

Introduction

What Is CSS?

- ▲ B2B and B2C solution
- ▲ Quick, easy and cost effective
- ▲ Built-in security
- ▲ Extensible product design
- ▲ Out-of-the-box functionality

What Is CSS?

Introduction

- CSS is QAD's B2B and B2C solution for web-enabling your Progress-based applications.
- CSS is the quickest, easiest, and most cost effective way to extend your Progress application to the web.
- Built into the CSS application is security and an extensible product design intended to maximize your current ERP implementation.
- Out-of-the-box functionality will quickly have your customer placing orders and reviewing real-time information within two weeks of installation.

Product Demonstration

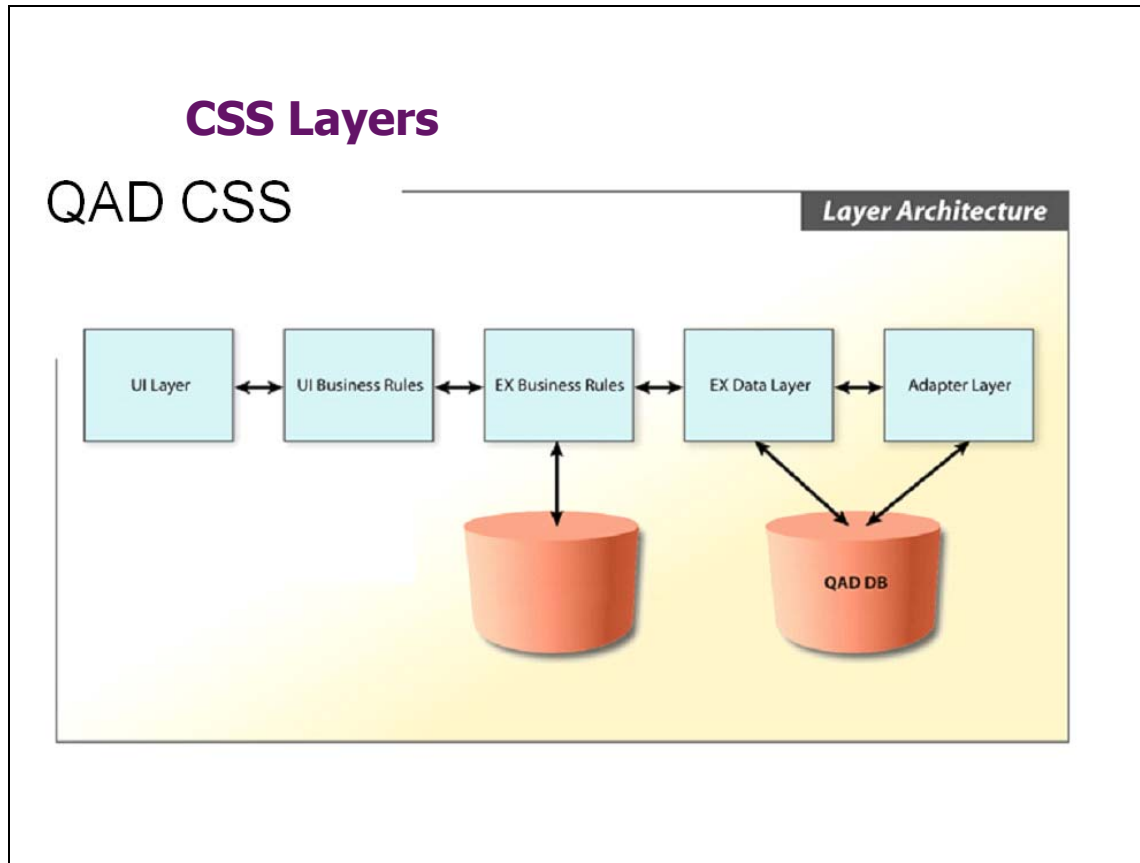
Product Demonstration

Course Overview

The concepts covered in this course are:

- ▲ Installation requirements of CSS
- ▲ Implementation procedures for CSS
- ▲ Architecture of CSS
- ▲ Customizing CSS
- ▲ Internationalization features in CSS

Course Overview



CSS Layers

CSS will be explored and explained in relationship to the base product. The purpose of the adaptor layer is to integrate CSS with existing ERP systems. Customization of this layer is covered in a separate course.

CSS Installation

Installation Topics covered:

- ▲ Installation guide
- ▲ Pre-trip questionnaire
- ▲ Installation fact Sheet
- ▲ System requirements
- ▲ Contents of CD
- ▲ Case study

CSS Installation

The installation chapter focuses on the requirements for installing the CSS base product. We will discuss:

- Installation Guide
- Installation fact sheet and its purpose
- Installation requirements that must be considered
- Contents of the installation CD
- Case study in which you will simulate a live site installation

CSS Implementation

CSS Administration Guide Review

- ▲ Creating security groups and users
- ▲ Menu modification
- ▲ Registry settings
- ▲ Item replication
- ▲ Mail events
- ▲ Case study

CSS Implementation

After you install the product at a customer's site, you have to initialize the product. In the chapter on implementation, we will review the CSS Admin Guide. We will review the instructions on how to create security groups and users. You will learn how to modify various menus and how to set up the CSS registry. You will also learn about item replication and setting up email events. At the end of this chapter, you will apply these site implementation tasks to your case study project.

CSS Architecture

Technologies applied:

- ▲ Persistence
- ▲ Procedure Overloading
- ▲ Preprocessors
- ▲ Dynamic Temp-Tables
- ▲ Dynamic Buffers
- ▲ Named Events
- ▲ Dynamic HTML (DHTML)
- ▲ CSS naming conventions

CSS Architecture

Technologies Applied

Before you as a developer can begin customizing CSS, you must understand the applied technologies and developed architecture that it utilizes.

We will review and discuss several advanced Progress 4GL coding techniques. These technologies include: persistence, procedure overloading, preprocessors, dynamic temp-tables and buffers, and named events. We will also review dynamic HTML (DHTML).

There is a standard naming convention used in CSS. We will explain that convention as it is applied to directories and files, variables and the database schema. The naming convention is also applied to user-defined functions, internal procedures and preprocessors.

Next, you will be introduced to the CSS layers and how they interact with each other. We will explain CSS' data architecture as well as the CSS superstructures or super-procedure libraries.

Customizing CSS

CSS customizable templates

- ▲ Basic template
- ▲ Maintenance template
- ▲ Report template

Customizing CSS

Customization techniques will be demonstrated in this section and then you will have the opportunity to apply the techniques to the case study. We will examine CSS templates that can be customized. These include:

- Basic template
- Maintenance template
- Report template

Customizing CSS

CSS Core API's

- ▲ Browse
- ▲ Calendar
- ▲ Event
- ▲ Session Management
- ▲ Registry
- ▲ Process Error
- ▲ Dataeng

Core APIs

You will also be introduced to the core APIs which you can use to add functionality to your newly created templates. These include:

- Browse
- Calendar
- Event
- Session Management
- Registry
- Process Error
- Dataeng

Internationalization

▲ Multi-lingual

Internationalization

The multi-lingual feature in CSS allows you to easily overcome communication barriers. This feature will be covered and you will get a chance to see how the various functions work.

Case Study Overview

- ▲ Create procedures from the basic template
- ▲ Link procedures together
- ▲ Create maintenance procedures from the maintenance templates
- ▲ Create report procedures from the report templates
- ▲ Use core objects
- ▲ Create and use new layers
- ▲ Modify the core product

Case Study Overview

CHAPTER 2

Installation

Installation

Before you begin:

- ▲ Installation Guide – what's in it?
- ▲ Pre-trip questionnaire
- ▲ Installation Planning Sheet
- ▲ System requirements
- ▲ Client/server compatibility
- ▲ Licensing information

Installation

Installation Guide

Pre-trip questionnaire

CSS pre-trip questionnaire: Webserver (messenger) Type: _____ OS: _____ Name: _____ IP: _____ Web Server Software (i.e., IIS, Apache, etc.): _____ Agent Box Type: _____ OS: _____ Name: _____ IP: _____ dB box Type: _____ OS: _____ Name: _____ IP: _____ MFG/PRO version: _____ Is CHUI installed: YES / NO Progress dB version: _____ Webspeed Version: _____ Have media: YES / NO Have Licenses: YES / NO Installed: YES / NO	Need Login for all boxes: WebServer Login Name _____ Password _____ Agent Login Name _____ Password _____ DB Login Name _____ Password _____ QAD login for CSS: Login Name _____ Password _____ Naming Conventions: _____ Setup Development / Production: Location of host logo: _____ Location of company color scheme: _____ Which OE Model: OE I OE II HDOE Dress code for consultant doing install: Suit Tie/Dress pants Business Casual Contact Names and info: _____ _____ _____
---	--

...but wait there's more!

Pre-Trip Questionnaire

This document is located in the appendix of the Installation Guide. Its purpose is to allow the implementer and customer an opportunity to discuss the installation. This pre-trip conversation should reveal and address any issues that need to be resolved prior to the implementer's arrival at the customer's site.

Installation Planning Sheet

Used to gather the following information:

- ▲ Where will the product be installed?
- ▲ What system rights and permissions will the installer have on the system?
- ▲ What TCP ports are available on the system?
- ▲ What web server software is being used?
- ▲ What version of Progress/WebSpeed is installed?
- ▲ Are the Progress/WebSpeed manuals available?
- ▲ What type of CSS distribution is this?

Installation Planning Sheet

Before you start the actual software install, you will need to fill out the Installation Fact Sheet. Its focus is the actual installation and environment decisions that need to be made at the time of the installation. It will address the following issues:

- Where will the product be installed?
- What system rights and permissions will the installer have on the system?
- What TCP ports are available on the system?
- What web server software is being used?
- What version of Progress/WebSpeed is installed?
- Are the Progress/WebSpeed manuals available?
- What type of CSS distribution is this?

System Requirements

Three areas:

- ▲ Hardware and Networking
- ▲ Java
- ▲ Disk Space

System Requirements

WebSpeed is configured to work with your system's hardware, networking configuration, and WebSpeed components. Your configuration might be on a single machine or it might be on a distributed network. If you already have WebSpeed set up and running, then you are almost there.

System requirements are broken down into the following areas:

- Hardware and networking
- Java
- Disk space

System Requirements

Hardware and Networking requirements:

Network access	Your systems must have access to an Intranet or the Internet
Network protocol	Your system must support the TCP/IP and UDP protocols. The platforms that WebSpeed currently supports provide built-in TCP/IP and UDP support.
Web server	Your Web server must support one of the following interfaces: ISAPI – For example, Microsoft Internet Information Server (IIS), versions 3.x and 4.x. NSAPI – For example, Netscape Enterprise or Fast Track Server, Version 3.x. CGI 1.1 – For example, Microsoft Internet Information Server (IIS), versions 3.x and 4.x; Apache; Netscape Enterprise or Fast Track Server, Version 3.x
Web browser	To use the new browser-based editor in WebSpeed version 3.1, you need one of the following Web browsers to run the WebSpeed Workshop: ➤ Microsoft Internet Explorer, version 5.0 or later. ➤ The version of Internet Explorer you are using must support JScript Version 2.0 ➤ Netscape Navigator, Version 4.5 or later.

Hardware and Networking

For CSS to function with your WebSpeed configuration, you need to pay special attention to four hardware and network components:

- Network access
- Network protocol
- Web server
- Web browser

System Requirements

- ▲ Java
 - The CSS product does not make any Java requirements beyond those of WebSpeed.
- ▲ Disk Space
 - Beyond the disk space requirements imposed by WebSpeed, CSS requires 100MB of free disk space.

Java and Disk Space

Client/Server Compatibility

- ▲ Because CSS requires WebSpeed version 3.1 for proper operation, the database server must be that of either Progress version 8.x or version 9.x.

Client/Server Compatibility

Licensing Information

- ▲ In the CSS product, the agents are tied to the database. They only allow agents to have access to the licensed modules. The number of licensed WebSpeed agents should be equal to the CSS user licenses.

Licensing Information

CD-Rom Contents

- ▲ Directories and files necessary for installing CSS
- ▲ Installation guides in PDF format for various installation platforms
- ▲ Administration guide in PDF format

CD-ROM Contents

On the CSS CD, you will find all the code necessary to perform an installation of the CSS product. You should begin by reading the corresponding PDF file of the Installation Guide for your platform.

CSS Directory Structure

- ▲ ad - admin
- ▲ adr - admin reports
- ▲ br - browse
- ▲ ca - customer access
- ▲ cc - credit card
- ▲ cl - clearance
- ▲ dom - distributed order management
- ▲ ext - extension / modification layer
- ▲ he - help
- ▲ lg - login
- ▲ lib - library
- ▲ messages - messages
- ▲ op - order process
- ▲ opr - order process reports
- ▲ qad - qad specific
- ▲ rep - replication
- ▲ so - sales order
- ▲ sys - system
- ▲ tools - tools
- ▲ tt - temp-tables
- ▲ xml - XML

CSS Directory Structure

Product Installation Demo

Product Installation Demo

CHAPTER 3

Implementation

Pre-Planning

- ▲ Before beginning the implementation processes, give special attention to the following:
 - How are the users to be organized?
 - What security groups should be created in response to the previous question?
 - What will be the role of the users who are placed in the various security groups?
 - What role variations (exceptions) might one or more users have from other users in the same security group?

Implementation

Pre-Planning

Why is pre-planning so important? What is the purpose of pre-planning? The built-in security features in CSS require some forethought on the part of the administrator prior to implementation.

Menus in CSS are generated for a user based on their security group. Security is applied regardless of whether or not the user used the menus. In other words, if a user bookmarks a particular screen in CSS, they cannot bypass the security.

User/customer relationships are assigned to specific security groups.

Product modules in CSS are assigned to one or more security groups. These factors result in each user being limited to specific modules for their user security group.

Security Group Maintenance

- ▲ Default settings
- ▲ Add new security groups
- ▲ Removing excess security groups

Security Group Maintenance

Security Group Maintenance allows CSS administrators to create groups of functions on a common menu. When a function is not assigned to a user's security group, the menu button or link will not show up on their screen. Administrators assign users to one security group thereby limiting users to specifically defined CSS modules. Security is enforced on access to every CSS page.

For more in-depth coverage, see the Security Group Maintenance section in the CSS Administration Guide.

User Maintenance

- ▲ Default settings
- ▲ Adding new users
- ▲ Removing excess users

User Maintenance

This function allows administrators to establish users and to create relationships between CSS user and ERP application “users.” User Maintenance is where personal IDs, passwords, contact information and preferences are available for changing and updating. Administrators have the ability to lock-out or add permissions specific to a user without losing important information that has already been input.

For more in-depth coverage, see the User Maintenance section in the CSS Administration Guide.

Setting Up Email Events

- ▲ Out-of-the-box features
- ▲ Initial settings
- ▲ Email recipients
- ▲ Email subject and body text replacements

Setting Up Email Events

Out of the box, CSS provides email events that allow notification of users or key personnel that need to be alerted when certain key events occur. CSS comes already set with many common email events.

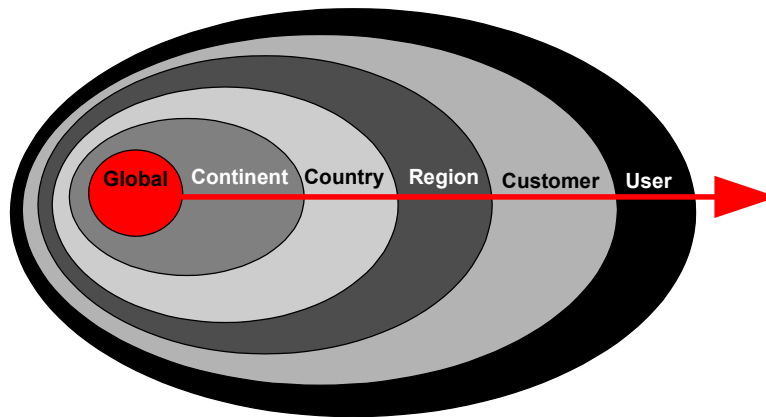
Initially, the email events are turned off in CSS. This is due to the fact that if any attempts are made to send email prior to setting up communications with the client's email server, the product will hang and result in a white screen of death (WSOD).

For each email event, you can attach one or more recipients. The recipients can be set up to always or conditionally receive email.

Both the subject and the body of the email message can contain replacement strings for more personalized or meaningful messages.

The CSS Registry

- ▲ What is it?
- ▲ Where is it used?



CSS Registry

What Is It?

The CSS system registry is a virtual control table that is comprised of keys and values. Like other control tables, administrators establish how CSS dynamically interprets and processes business rules. Unlike other control tables, the system registry allows an administrator to tailor CSS functionality for specific users or groups of users.

Where Is It Used?

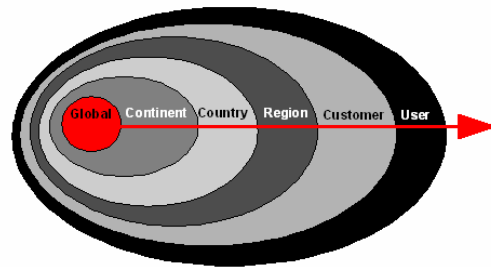
It is used throughout the CSS product. It serves as the foundation for most CSS business rules. It is through the application of the registry that the flexibility of CSS is demonstrated.

The registry is designed to allow for n-th degree of customization of any registry key.

The CSS Registry: How Does It Work?

wpro_registry

Version	"ex4.0"
Module	"OP"
Id	<i>group_id</i>
Key	"CreditHold"
Value	"Yes"
Override	<i>logical</i>
Usage	<i>Description</i>



Registry Key Descriptor: (C1,C2,Cn),(M1,M2,Mn),(S),(N),(U)

C = Customer Groups
M = Marketing Groups
S = Security Groups
N = Customer Number
U = User Id

How Does It Work?

The CSS registry is an assembly of key value pairs. What makes it different from most control tables is the existence of an additional field: ID

When CSS needs to obtain the value for a specific key, it first collects all the groups that the current user is associated with. It then recursively evaluates all the matching keys against each associated group to arrive at the final value of the lowest key.

For convenience and organization, the key is extended to include a module and version descriptor. This allows you to organize registry keys into logical or functional areas.

For optimization purposes, an override field is provided to limit the recursive nature of the evaluation of the ID values.

46 MFG/PRO TRAINING GUIDE — QAD CSS (CUSTOMER SELF SERVICE): DEVELOPER'S COURSE

The registry also contains a usage field that provides for documentation of the key. It should include valid values for the key.

The CSS Registry

The following registry flags may need to be changed during implementation:

compLogoHref	jsmsgdir
compLogoName	lmenustartMenuX
hostCopyrightHref	lmenustartMenuY
hostCopyrightText	menuDir
hostURL	OEHome

Many more flags are explained in the Registry Flag Appendix in the Administration Guide

Common Flags

Lab Exercise

- ▲ Do the CSS Registry Lab for Chapter 3 Lab Exercises found in Appendix A

Lab Exercise

Item and Customer Replications

- ▲ What is it?
- ▲ What is its purpose?
- ▲ How does it work?

Item and Customer Replications

CHAPTER 4

CSS Layers

CSS Layers

- ▲ UI layer
- ▲ UI Business Rules
- ▲ EX Business Rules
- ▲ EX Data Layer
- ▲ Adapter Layer

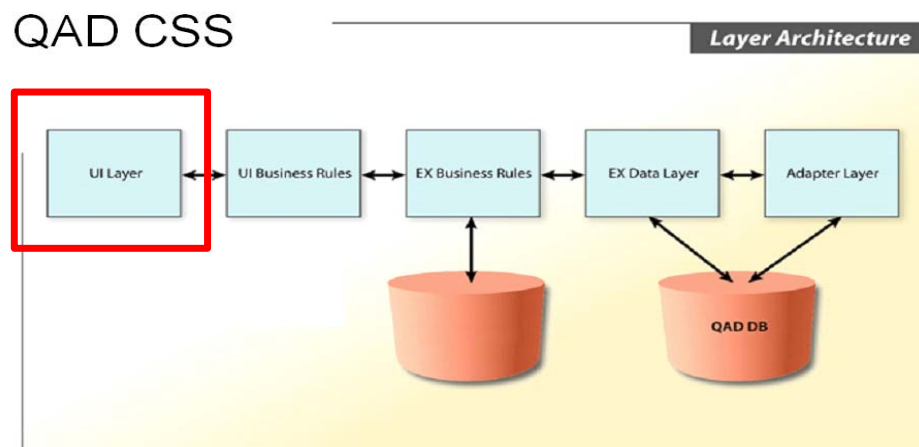
CSS Layers

Overview

The CSS product is comprised of five layers. Each layer has specific responsibilities for handling various aspects of the application. The five layers are the UI layer, the UI Business Rules layer, the EX Business Rules layer, the Data layer, and the Adapter layer. Each layer is divided up based on what functional services it provides and the technology used in creating that particular layer.

UI Layer

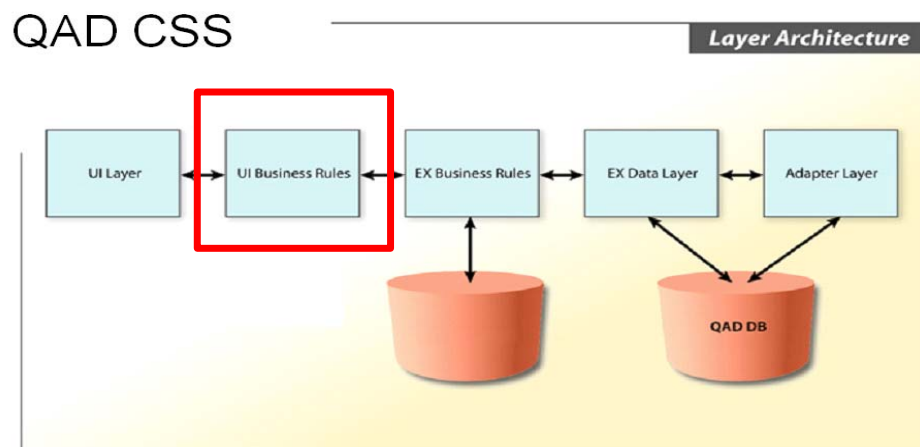
QAD CSS



UI Layer

This layer is responsible for the presentation and collection of information that interacts with the user interface. It is written using HTML, Progress SpeedScript, and Javascript. Its purpose is to serve as the container for specific user areas of dynamic content controlled by the UI Business Rules layer.

UI Business Rules Layer

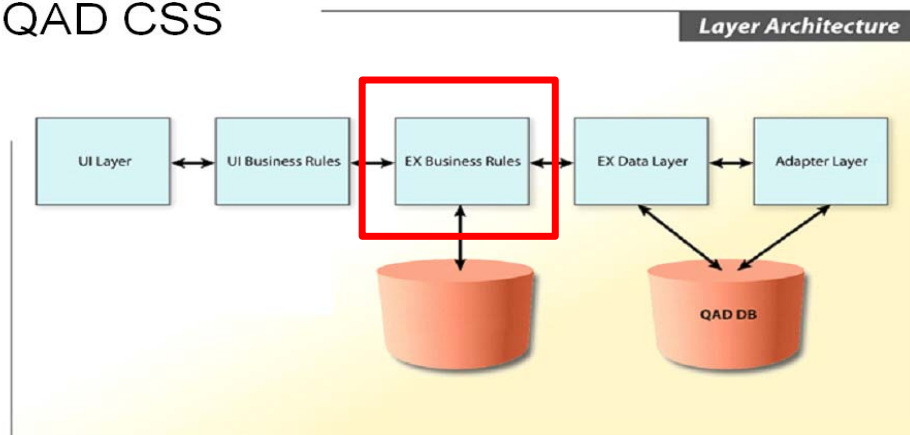


UI Business Rules Layer

This layer is responsible for determining and generating the dynamic content of CSS. It is accomplished by interacting with the UI layer and the EX Business Rules layer. It formats the dynamic content.

EX Business Rules Layer

QAD CSS



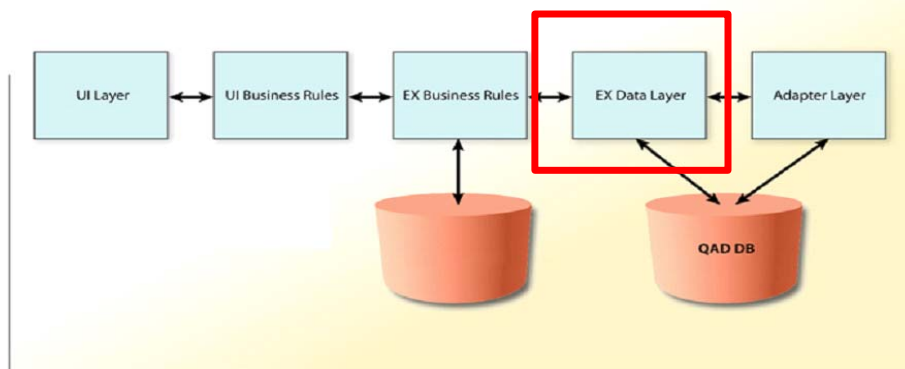
EX Business Rules Layer

This layer serves as the central hub for all information that is processed through CSS. It interacts with the UI Business Rules layer and EX Data layer using a series of APIs that pass information and directives between the two layers. To satisfy these requests, the EX Business Rules layer interacts with the EX Data layer, Adapter layer and the CSS database.

EX Data Layer

QAD CSS

Layer Architecture

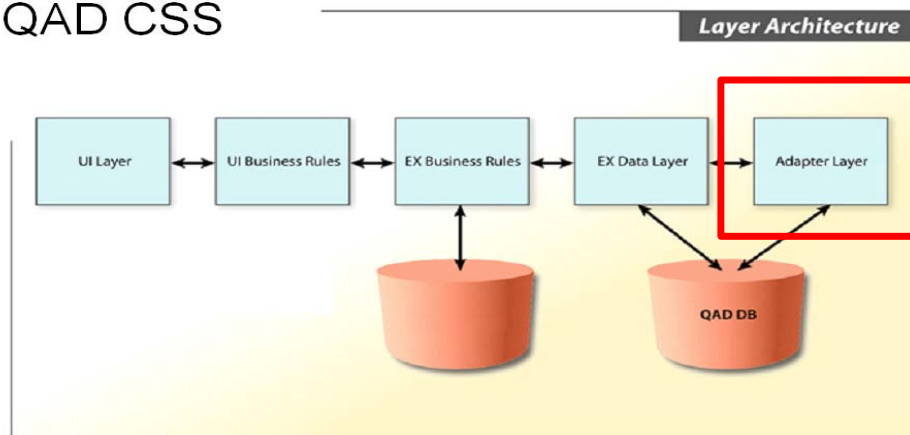


EX Data Layer

This layer provides a set of centralized procedures for retrieving and navigating through sets of records. The EX Data layer also provides common procedures for updating the database with the necessary change control processes that are required to maintain stateless applications.

Adapter Layer

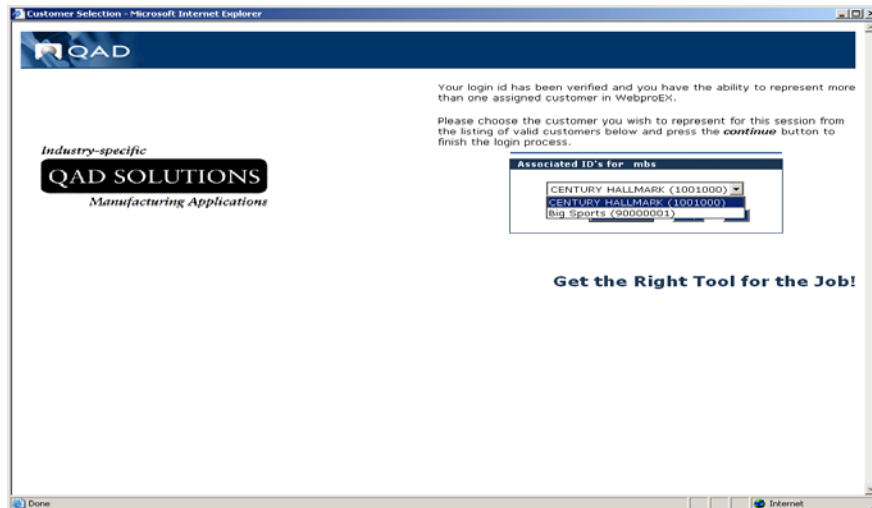
QAD CSS



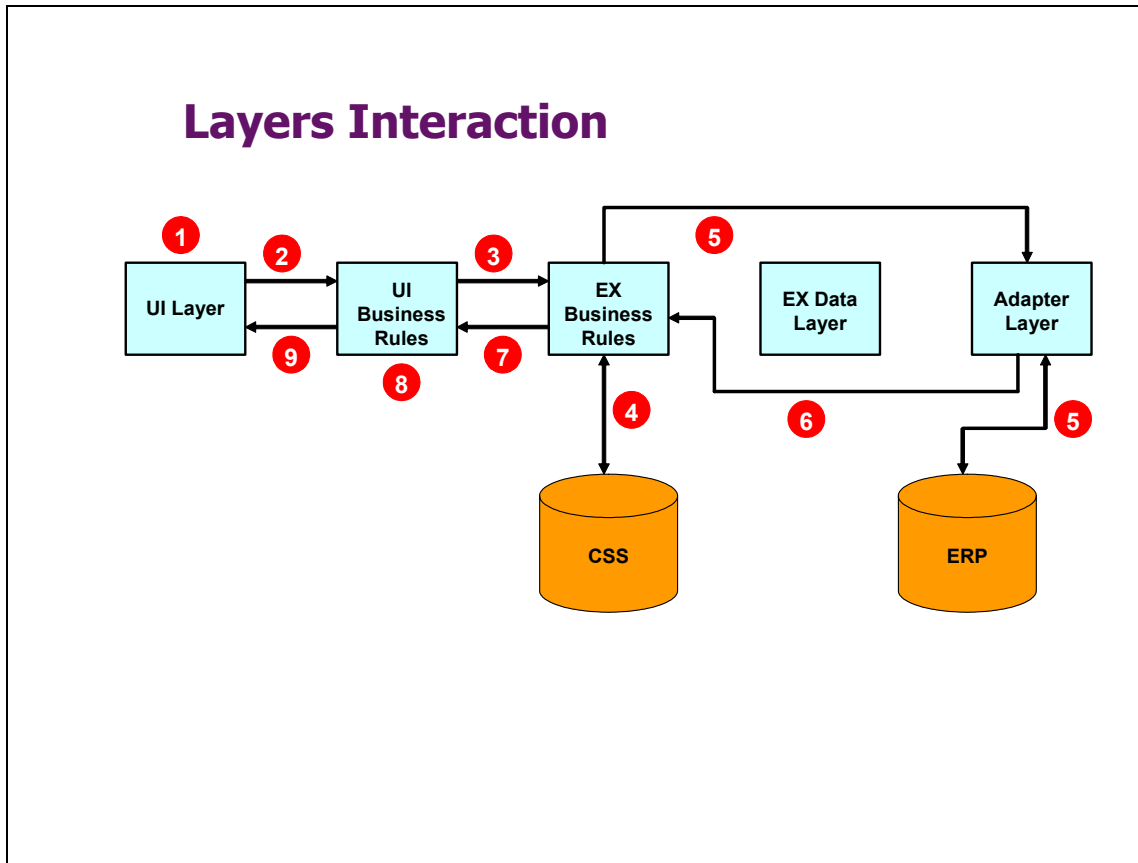
Adapter Layer

This final layer is responsible for all communications between the CSS Business Rules layer and the ERP application. All data access must be performed in the Adapter layer. It also serves as a bridge to translate information requests from CSS APIs into those APIs understood by the ERP application.

Layers Interaction



Layers Interaction

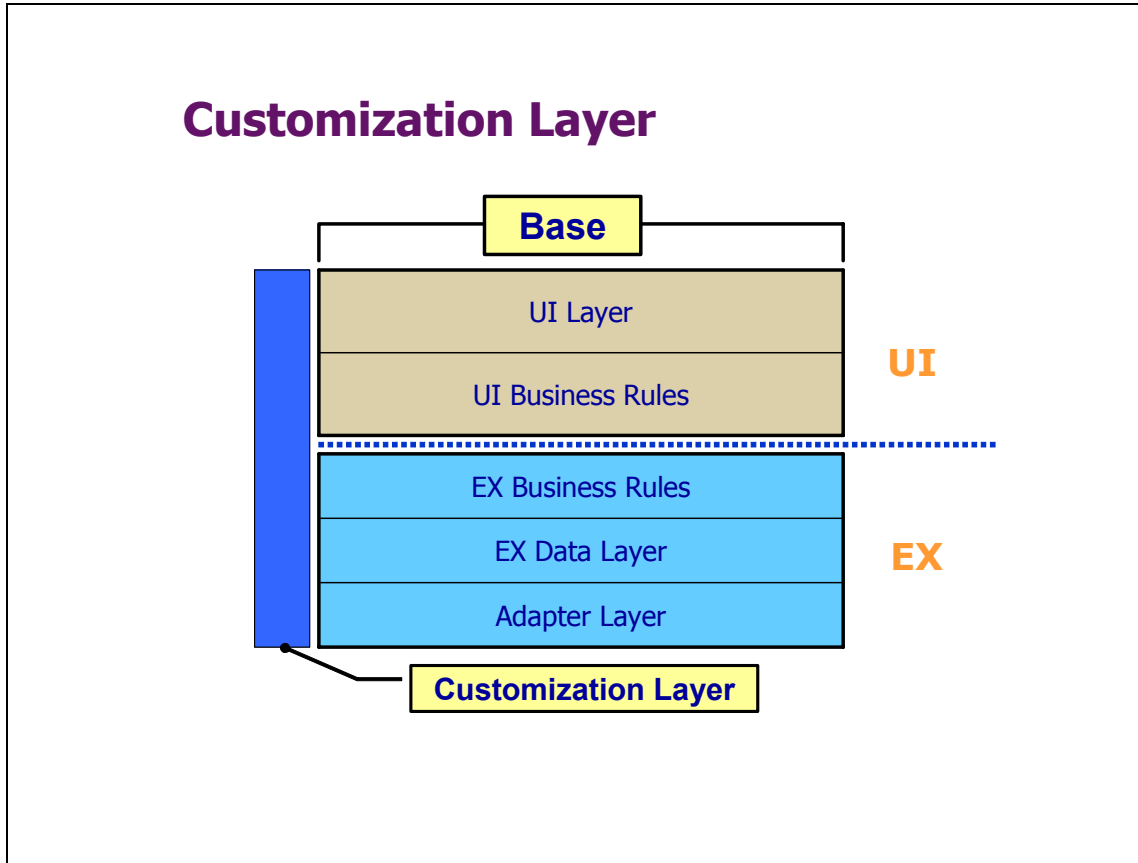


Layers Interaction, cont.

- 1 UI layer gathers and loads container, styles, Javascript libraries, and specific text
- 2 Call is made to UI Business Rules
- 3 Call is made to EX Business Rules
- 4 Get user associations for login from CSS database
- 5 Get customer name from ERP database via Adapter layer
- 6 Return correct customer list to EX Business Rules layer
- 7 Return list to UI Business Rules layer

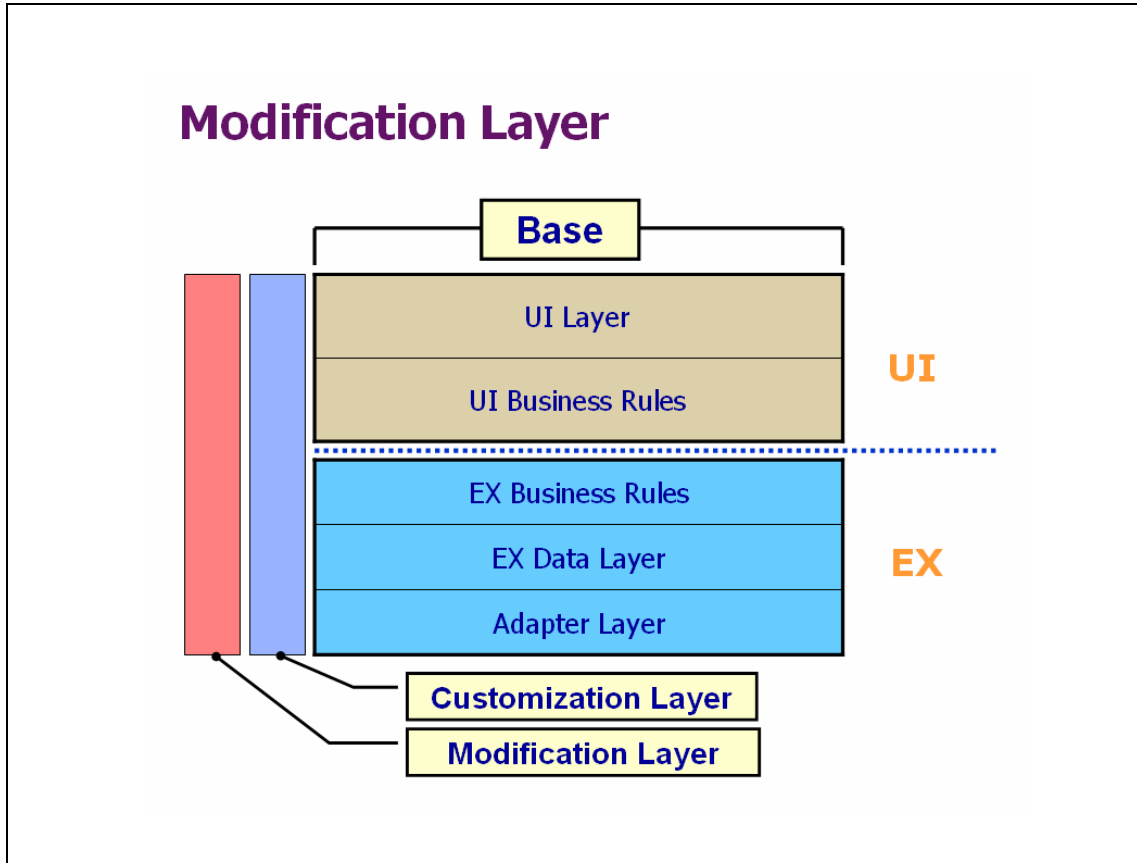
60 MFG/PRO TRAINING GUIDE — QAD CSS (CUSTOMER SELF SERVICE): DEVELOPER'S COURSE

- 8** Format return customer list into selection list object
- 9** Return control to UI layer



Customization Layer

This layer provides a level of customization apart from the base product. It allows the developer the ability to change settings within the product without requiring modifications to the base product. This is achieved through the use of the CSS Registry.



Modification Layer

This layer provides for the modification of any CSS internal procedure or function without requiring access to the base product. This is achieved through the use of Progress Super Procedures.

CHAPTER 5

Technologies Applied

Technologies Applied

Progress 4GL

- ▲ Preprocessors
- ▲ Internal Procedures
- ▲ User-Defined Functions
- ▲ Temp-Tables
- ▲ Persistence
- ▲ Super Procedures
- ▲ Dynamic Objects
- ▲ Named Events

Introduction

Progress 4GL

In this chapter, we will review the Progress 4GL technologies or coding techniques that are applied in the CSS product. We will use the Progress procedures editor and Appbuilder in order to simplify looking at the code.

Technologies Applied

Web technologies

- ▲ Dynamic HTML
 - HTML
 - JavaScript
 - Style sheets

Web Technologies

Preprocessors

- ▲ The Preprocessor is included as part of the PROGRESS compiler.
- ▲ The Preprocessor allows you to perform the following:
 - Setup substitution parameters (preprocessor names), which are resolved prior to compilation throughout a procedure.
 - Conditionally compile parts of a procedure.
- ▲ The Preprocessor uses compiler directives placed throughout your code. Directives start with an ampersand (&) and are only meaningful to the preprocessor.

Preprocessors

Preprocessors

```

/* reorder1.p */
&GLOBAL-DEFINE FRAME-COLOR 14

FOR EACH customer:
  {examples\item.dsp
    &FIELD1="custnum"
    &FIELD2="name"}

  {examples\item.dsp
    &FIELD1="city"
    &FIELD2="state"
    &FIELD3="country"}

  &UNDEFINE FRAME-COLOR

  {examples\item.dsp
    &FIELD1="salesRep"
    &FIELD2="contact"
    &FIELD3="phone"}

END.

```

```

/* item.dsp */

&SCOPED-DEFINE FRAME-NAME f-name{&SEQUENCE}

DISPLAY
  {&FIELD1}
  {&FIELD2}
  {&FIELD3}
WITH &IF DEFINED(FRAME-COLOR) <> 0 &THEN
  BGCOLOR {&FRAME-COLOR} &ENDIF
  FRAME {&FRAME-NAME} CENTERED
  TITLE "FRAME {&FRAME-NAME}"

```

Global vs Scoped

Use GLOBAL/SCOPE directives to define a preprocessor name.

Syntax

```

&GLOBAL-DEFINE preprocessor-name definition
&GLOBAL-DEFINE preprocessor-name definition

```

Where preprocessor-name is the unique name of the substantial parameter, definition is the character string substituted for the preprocessor name prior to compilation.

Use {&preprocessor-name} to reference a previously defined preprocessor name. A preprocessor name has a specific scope. A preprocessor name defined with &GLOBAL-DEFINE remains in effect from the point where it was defined to the end of the program being compiled. An &SCOPED-DEFINED preprocessor name is available for use after it has been defined and only

within the file in which it is defined. Use an `&UNDEFINE` directive to turn off a preprocessor name. Redefining a GLOBAL-DEFINE preprocessor name in an include generates a warning and results in the value changing for the remainder of the program.

```

/* reorder1.p */
&GLOBAL-DEFINE FRAME-COLOR 14

FOR EACH customer:
  {examples\item.dsp
    &FIELD1="custnum"
    &FIELD2="name"}

  {examples\item.dsp
    &FIELD1="city"
    &FIELD2="state"
    &FIELD3="country"}

  &UNDEFINE FRAME-COLOR

  {examples\item.dsp
    &FIELD1="salesRep"
    &FIELD2="contact"
    &FIELD3="phone"}
END.

```

```

/* item.dsp */

&SCOPED-DEFINE FRAME-NAME f-name{&SEQUENCE}

DISPLAY
  {&FIELD1}
  {&FIELD2}
  {&FIELD3}
WITH &IF DEFINED(FRAME-COLOR) <> 0 &THEN
  BGCOLOR {&FRAME-COLOR} &ENDIF
FRAME {&FRAME-NAME} CENTERED
TITLE "FRAME {&FRAME-NAME}" .

```

Preprocessor Example

FRAME f-name0		
Cust Num	Name	
3	Hoops	

FRAME f-name1		
City	State	Country
Atlanta	GA	USA

FRAME f-name2		
Sales Rep	Contact	Phone
HXM	Michael Traitser	(617) 355-1557

Press space bar to continue.

Preprocessor Example

Preprocessors

Conditional Compilation

- &IF *expression* &THEN
- &ELSEIF *expression* &THEN
- &ELSE
- &ENDIF

Predefined Preprocessor	Description
{&OPSYS}	Operating system name
{&FILE-NAME}	Contains the name of the procedure being compiled.
{&LINE-NUMBER}	Returns the current line number being compiled. If placed within an include file, the line number returned is relative to the beginning of the include file.
{&SEQUENCE}	Returns an integer value that is incremented each time the directive is referenced.
{&WINDOW-SYSTEM}	Returns the name of the windowing system PROGRESS is running under.
{&MESSAGE}	Allows you to display a character string at compile time.
{&BATCH-MODE}	Returns "yes" if the -b startup parameter was used to start the client session.

Conditional Compilation

Use the &IF, &THEN, and &ELSE directives to conditionally include blocks of code in a compiled procedure.

Expressions can contain:

- Other preprocessor names
- Operators
- Preprocessor functions (See Programmers Handbook, page 8-9)
- DEFINED() preprocessor function (See Programmers Handbook, page 8-4)

Progress automatically provides several preprocessor names. They can be referenced anywhere within your source code, including preprocessor expressions. They are read-only; they cannot be redefined.

Preprocessors

- ▲ There are several preprocessors available to you throughout the CSS product:
 - {&ON-UI}
 - {&ON-EX}
 - {&ON-SX}
 - {&ON-DB}
 - {&PS}
 - {&HTML-HIDDEN-FIELDS}
 - {&ORDER-HIDDEN-FIELDS}
 - {&EXD}
 - {&EXDNC}
 - {&EXDEBUG}

Preprocessors in CSS

Internal Procedures

- ▲ An internal procedure is a block of PROGRESS code contained within a larger PROGRESS procedure.
- ▲ You reference an internal procedure by issuing a RUN statement.
- ▲ Internal procedures allow you to modularize your code and reduce the number of external files in an application.
- ▲ All resources scoped to the main procedure are available within the internal procedure.
- ▲ A single external procedure (.p or .w) can contain any number of internal procedures.

Syntax

```
PROCEDURE proc-name: [4GL Statements] END.
```

Code Example

```
/* custbal.p */
DEF VAR v-disc-bal LIKE customer.balance LABEL "Discounted Balance".
DEF VAR vUpdateCust AS LOGICAL NO-UNDO.
REPEAT:
  PROMPT-FOR customer.custNum.
  FIND customer USING customer.custNum.
  DISPLAY customer WITH 2 COL 1 DOWN.
  RUN calc-disc.ip.
  MESSAGE "Update this customer?" UPDATE vUpdateCust.
  IF vUpdateCust THEN
    DO TRANSACTION:
      UPDATE customer.balance.
      RUN calc-disc.ip.
  END.
END.
PROCEDURE calc-disc.ip:
  v-disc-bal = customer.balance * (customer.discount / 100).
  DISPLAY v-disc-bal WITH FRAME f1 ROW 13 CENTERED SIDE-LABELS
  TITLE "DISCOUNTED BALANCE".
END. /* calc-disc */
```

User-Defined Functions

- ▲ Like internal procedures, user defined functions provide a container for application logic.
- ▲ Access to user defined functions is not provided via a RUN statement, but rather as any other 4GL function.
- ▲ User Defined Functions always return some value.

Syntax

```
FUNCTION function-name [RETURNS] data-type [(param [, param
] . . . )][FORWARD | [MAP [TO] actual-name]IN proc-handle]
```

Code Example, cont.

```
/* func1.p */
FUNCTION fRemainCredit RETURNS DECIMAL (INPUT vCust# AS INTEGER):
  FIND customer WHERE customer.custNum = vCust# NO-LOCK NO-ERROR.
  IF NOT AVAIL customer THEN
    MESSAGE "Remaining credit cannot be calculated for a non-existent
customer."
    VIEW-AS ALERT-BOX.
  RETURN (customer.creditLimit - customer.balance).
END.

FOR EACH order NO-LOCK:
  DISPLAY order.orderNum
    order.orderDate
    order.custNum
    fRemainCredit(order.custNum)
    COLUMN-LABEL "Remaining!Credit" .
END.
```

Temp-Tables

- ▲ Actually database tables stored in a temporary database.
- ▲ Stored on disk instead of memory.
- ▲ May contain multiple indexes.
- ▲ The -Bt parameter allows the user to specify the buffer pool size for temporary tables

Syntax

```
DEFINE [ [ NEW ] SHARED ] TEMP-TABLE temp-table-name
[NO-UNDO ] [ LIKE table-name [ USE-INDEX index-name [ AS
PRIMARY ] ] ... ] [ FIELD field-name { AS data-type } [
field-options ] ] ... { LIKE field } INDEX index-name [IS
[ UNIQUE ] [ PRIMARY ] ] {index-field [ ASC ] } ... [DESC]
```

Temp-Tables

Temporary tables are actually database tables stored in a temporary database. They are not entirely RAM resident, which means their size is limited only by the size of your disk. You reference a temporary table the same way you reference a database table. They only last for the duration of the procedure that creates them (unless they are shared) and are not shared among application users.

The table itself is set up in the directory where Progress stores temporary files. This is usually the directory from which you enter the database. You can control the placement of temporary files with the -T startup option. Temporary Tables are named with a DBI prefix followed by a process ID.

The -Bt parameter allows the user to specify the buffer pool size for temporary tables. This parameter is logically identical to the -B parameter controlling the buffer pool size for databases. Additional buffers can be created for temporary tables.

Temporary tables can contain multiple indexes, multi-component indexes, and unique indexes.

Persistence

- ▲ Two types of procedures:
 - Non-persistent (default)
 - Persistent (explicitly defined)
- ▲ Non-persistent Procedure Scope
 - begins when procedure is executed via a RUN statement
 - ends when control returns to the calling program
- ▲ Persistent Procedure Scope
 - begins when procedure is executed via a RUN statement
 - optional PERSISTENT phrase is invoked
 - exist for the remainder of PROGRESS session or until they are explicitly deleted.

Persistence

External Progress procedures come in two types: non-persistent, which is the default, and persistent, which must be explicitly defined.

Non-persistent procedures are created (come into scope) when they are executed via a RUN statement. They cease to exist (out of scope) when control returns to the calling program after the RUN statement.

Persistent procedures provide an extremely effective means of making your application modular. One commonly used technique is to place key internal procedures inside a persistent procedure. Once the persistent procedure is brought into scope, then any of its internal procedures can be repeatedly executed from anywhere within an application. In addition to promoting modularity, using persistent procedures contributes to smaller Progress executables (R-code).

Syntax

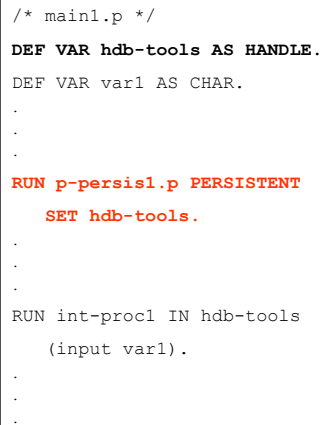
```
RUN {extern-procedure | VALUE(extern-expression) | path-  
name<member-name>} [PERSISTENT [SET handle]] [parameter[ ,  
parameter] . . . )]
```

Persistence

```

/* main1.p */
DEF VAR hdb-tools AS HANDLE.
DEF VAR var1 AS CHAR.
.
.
.
RUN p-persis1.p PERSISTENT
  SET hdb-tools.
.
.
.
RUN int-proc1 IN hdb-tools
  (input var1).
.
.
.

```



```

/* p-persis1.p */

PROCEDURE int-proc1:
  DEF INPUT PARAM v1 AS CHAR.
  [procedure logic]
END.

PROCEDURE int-proc2:
  [procedure logic]
END.

```

Persistent Option

Persistent procedures are created (come into scope) when they are executed via a RUN statement. They continue to exist for the remainder of the Progress session or until they are explicitly deleted. To execute a procedure persistently, specify PERSISTENT on the RUN statement.

Multiple instances (occurrences) of the same persistent procedure are created by executing the procedure many times via the RUN statement.

To access an internal procedure contained within a persistent procedure, you must attach a handle to the containing procedure. Use the SET handle parameter of the RUN statement to attach the handle to the procedure.

Example

```
RUN pgm0.p PERSISTENT SET vhandle.
```

Persistence

```

/* main1.p */
DEF VAR hdb-tools AS HANDLE.
DEF VAR var1 AS CHAR.
.
.
.
RUN p-persis1.p PERSISTENT
    SET hdb-tools.
.
.
.
RUN int-proc1 IN hdb-tools
    (input var1).
.
.
.

```

```

/* p-persis1.p */

PROCEDURE int-proc1:
    DEF INPUT PARAM v1 AS CHAR.
    [procedure logic]
END.

PROCEDURE int-proc2:
    [procedure logic]
END.

```

Procedure Handles

Use the IN handle parameter of the RUN statement to indicate the container for the internal procedure. Effectively, the handle specified is a pointer to the procedure that contains the internal procedure you are trying to execute.

Run pgml.ip IN vhandle.

Syntax

```

RUN {intern-procedure|VALUE(intern-expression)}
[IN handle] [(parameter[ , parameter]...)] [NO-
ERROR]

```

Persistence

The handle to a procedure has many useful attributes:

Attribute	Description
DB-REFERENCES	Comma delimited list of the logical databases names referenced in the procedure.
FILE-NAME	Full path of the current procedure.
GET-SIGNATURE	The type and parameters of an internal procedure.
INTERNAL-ENTRIES	Comma delimited list of internal procedures contained within the current procedure.
PERSISTENT	True/ False condition regarding persistent status of current procedure.
PRIVATE-DATA	Character string used to provide information about the current procedure.

Persistent Attributes

The handle to a persistent procedure can be stored in a variable and used to access the properties of the procedure. THIS-PROCEDURE is a system handle that can be used within a procedure to access its attributes.

Persistence Example

ppord1.p

The screenshot shows three overlapping windows from a legacy application:

- Customer Orders** (Top): A table with columns 'Order Num', 'Ordered', 'Carrier', and 'Shipped'. It contains 8 rows of order data.
- Order Lines** (Bottom Left): A table with columns 'Line Num', 'Item Name', and 'Qty'. It contains 6 rows of item details.
- Customer Details** (Bottom Right): A form displaying customer information for 'Cust Num: 53'.

The **Customer Orders** window has buttons for 'Order Lines', 'Customer', and 'Exit'. The **Order Lines** window has an 'Exit' button. The **Customer Details** window has an 'OK' button.

Customer Orders Data:

Order Num	Ordered	Carrier	Shipped
1	01/26/98	FlyByNight Courier	01/31/1998
2	10/15/97	Walkers Delivery	10/10/1997
3	09/23/97	Standard Mail	09/28/1997
4	01/17/98	Standard Mail	01/22/1998
5	02/12/98	Standard Mail	02/17/1998
6	02/12/98	Standard Mail	02/16/1998
7	03/06/98	Standard Mail	03/11/1998
8	09/18/97	Standard Mail	09/23/1997

Order Lines Data:

Line Num	Item Name	Qty
1	Canoe	31
2	Surfboard	21
3	Swimming Trunks	77
4	Ski Gloves	39
5	Baseball	75
6	Catchers Mit	28

Customer Details Data:

Cust Num:	53
Name:	Offside Hockey
Contact:	Sven Tumba
Balance:	6,275.17
Discount:	10%
Credit Limit:	15,000

ppdet1.p

Code Demonstration

- ppord1.p
- ppdet1.p

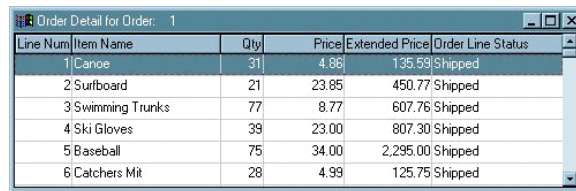
Persistence Example



Order Num	Name	Ordered	Shipped	Order Status
1	Offside Hockey	01/26/98	01/31/1998	Shipped
2	Off The Wall	10/15/97	10/10/1997	Shipped
3	First Down Football	09/23/97	09/28/1997	Shipped
4	Fallen Arch Running	01/17/98	01/22/1998	Shipped
5	Birdy's Badminton	02/12/98	02/17/1998	Shipped
6	Lift Tours Inc.	02/12/98	02/16/1998	Shipped
7	Butternut Squash Inc	03/06/98	03/11/1998	Shipped
8	Hoopelin Pesapallo	09/18/97	09/23/1997	Shipped

w-order1.w

w-olines1.w



Line Num	Item Name	Qty	Price	Extended Price	Order Line Status
1	Canoe	31	4.86	135.59	Shipped
2	Surfboard	21	23.85	450.77	Shipped
3	Swimming Trunks	77	8.77	607.76	Shipped
4	Ski Gloves	39	23.00	807.30	Shipped
5	Baseball	75	34.00	2,295.00	Shipped
6	Catchers Mit	28	4.99	125.75	Shipped

Code Demonstration, cont.

- w-order1.w
- w-olines1.w

Super Procedures

- ▲ Super procedures provide highly flexible re-usable components by:
 - Providing easy access to standard functionality
 - Providing the ability to override or enhance standard functionality without changing the standard functionality.
- ▲ When coded properly custom functionality will not be effected by new releases of standard functionality.

Super Procedures

Introduction

Progress V9 provides 4GL methods for override and overloading internal procedures and user-defined functions. This coding technique allows implicit access to internal procedures and user-defined functions independent of where they are physically located. Developers no longer need to supply the handle of the procedure containing the function or internal procedure being called. This type of procedure is known as a Super Procedure.

Super Procedures form a large part of the underlying methodology used in the ADM2. In fact, Super Procedures have eliminated the need for the Dispatch Event Procedure.

Super Procedures

```

/* notsuper.p */
DEFINE VAR vFunc_hdl AS HANDLE.
DEFINE VAR vCreditLimit LIKE Customer.CreditLimit LABEL "Credit Limit".
DEFINE VAR vType AS CHAR FORMAT "x(10)" LABEL "Type".
CURRENT-WINDOW:WIDTH = 100.

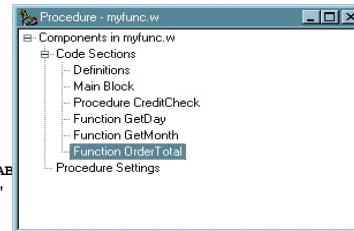
FUNCTION GetMonth RETURNS CHAR (INPUT vDate AS DATE) IN vFunc_hdl.
FUNCTION GetDay RETURNS CHAR (INPUT vDate AS DATE) IN vFunc_hdl.
FUNCTION OrderTotal RETURNS DECIMAL (INPUT vOrderNum AS INT) IN vFunc_hdl.

RUN myfunc.w PERSISTENT SET vFunc_hdl.

FOR EACH Order NO-LOCK:
  RUN CreditCheck IN vFunc_hdl
  (INPUT Order.OrderNum, OUTPUT vCreditLimit, OUTPUT vType).

  DISPLAY Order.OrderNum
  STRING(GetDay(Order.OrderDate)) + ", " +
  STRING(GetMonth(Order.OrderDate)) + " " +
  STRING(DAY(Order.OrderDate)) FORMAT "x(25)" COLUMN-LAB
  OrderTotal(Order.OrderNum) COLUMN-LABEL "Order Total"
  vType
  vCreditLimit WITH WIDTH 100 THREE-D USE-TEXT.
END.

```



Called Procedure Example

To access User-Defined Functions or Internal Procedures contained within a persistent procedure, a developer must maintain a handle of the persistent procedure.

Code Demonstration

nonsuper.p

Super Procedures

▲ ADD-SUPER-PROCEDURE() Method

- This method binds a procedure to another procedure or session. Internal Procedures and User-Defined Functions can be accessed as though they were defined in the target procedure.

ADD-SUPER-PROCEDURE()

Syntax

ADD-SUPER-PROCEDURE (super-proc-hdl[,proc-search])

super-proc-hdl

The handle of a running persistent procedure that you want to make a super procedure of the local procedure or of the current Progress session.

proc-search

Optional expression that determines which super procedures are searched when super-proc-hdl invokes RUN SUPER or the SUPER function.

Super Procedures

```

/* gosuper.p */
DEFINE VAR vFunc_hdl AS HANDLE.
DEFINE VAR vCreditLimit LIKE Customer.CreditLimit LABEL "Credit Limit".
DEFINE VAR vType AS CHAR FORMAT "x(10)" LABEL "Type".
CURRENT-WINDOW:WIDTH = 100.
/* Forward declarations generated by ProtoGen Tool */
{myfuprto.i}

RUN myfunc.w PERSISTENT SET vFunc_hdl.

THIS-PROCEDURE:ADD-SUPER-PROCEDURE(vFunc_hdl).

FOR EACH Order NO-LOCK:
  RUN CreditCheck (INPUT Order.OrderNum, OUTPUT vCreditLimit, OUTPUT vType).

  DISPLAY Order.OrderNum
    STRING(GetDay(Order.OrderDate)) + ", " +
    STRING(GetMonth(Order.OrderDate)) + " " +
    STRING(DAY(Order.OrderDate)) FORMAT "x(25)" COLUMN-LABEL "Order Date"
    OrderTotal(Order.OrderNum) COLUMN-LABEL "Order Total"
    vType
    vCreditLimit WITH WIDTH 100 THREE-D USE-TEXT.
END.

```

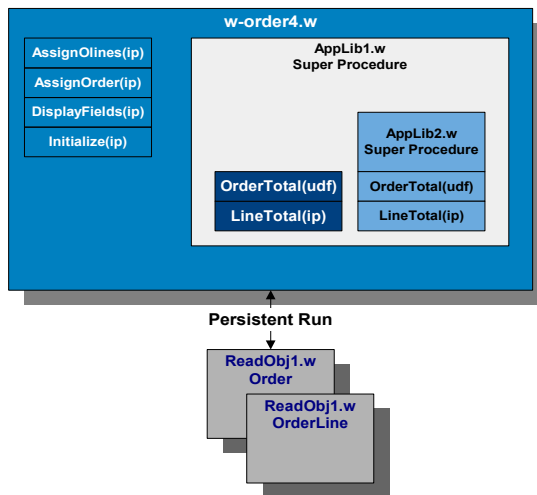
Super Procedure Example

The ADD-SUPER-PROCEDURE method binds myfunc.w to gosuper.w. Once myfunc.w is bound to gosuper.p User-Defined Functions and Internal Procedures can be accessed as though they are part of gosuper.p.

Code Example

gosuper.p

Super Procedures



Super Procedures, cont.

Super Procedures

w-order4.w

The screenshot shows a window titled "Orders" with a standard Windows-style title bar. The window is divided into several sections for data entry and navigation.

Orders Section:

- Order Num: 3
- Cust Num: 66
- Ordered: 09/23/97
- Carrier: Standard Mail
- Promised: 09/28/97
- Shipped: 09/28/1997
- Credit Card: American Express (dropdown menu)
- Order Total: \$23,519.59

Navigation Buttons: First, Prev, Next (highlighted), Last, Save, Done.

OrderLines Section:

- Order Num: 3
- Line Num: 1
- Item Num: 45
- Order Line Status: Shipped (dropdown menu)

Item Details:

- Qty: 95
- Price: 81.00
- Discount: 5%
- Extended Price: 7,310.25

Buttons: < > (navigation), Save, Calculate.

Business Logic

The screen capture displayed above is of a table maintenance program, which uses separate components containing business logic. The business logic is implemented using super procedures.

Code Demonstration

- w-order4.w
- AppLib1.w
- ReadObj1.w

Super Procedures

w-order4.w

```

MAIN-BLOCK:
DO ON ERROR UNDO MAIN-BLOCK, LEAVE MAIN-BLOCK
ON END-KEY UNDO MAIN-BLOCK, LEAVE MAIN-BLOCK:
RUN enable_UI.

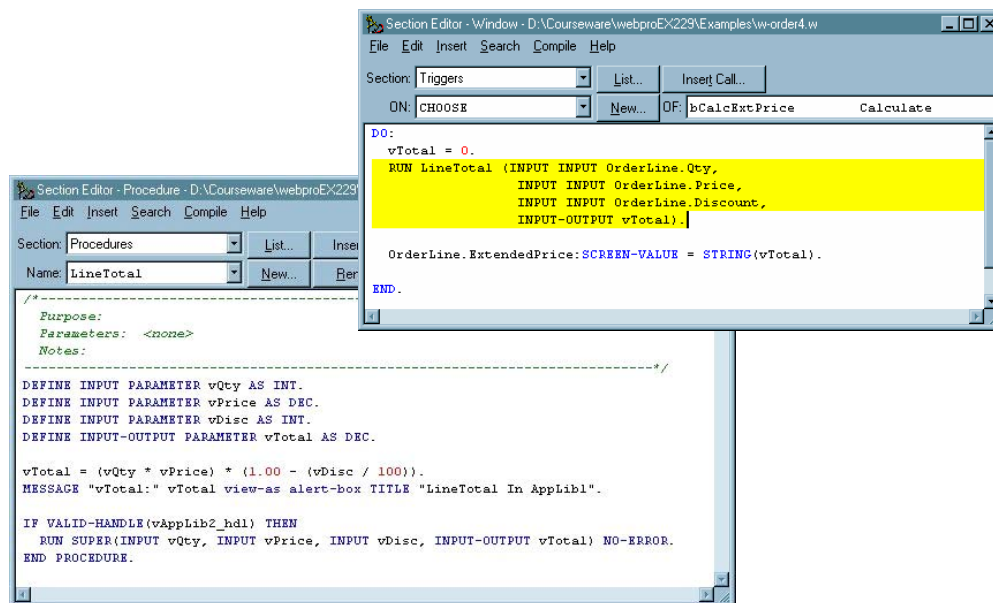
/* Bind Super Procedure */
IF NOT VALID-HANDLE(vAppLib_hdl) THEN
  RUN AppLib1.w PERSISTENT SET vAppLib_hdl.
IF NOT THIS-PROCEDURE:ADD-SUPER-PROCEDURE(vAppLib_hdl) THEN
  MESSAGE 'Could not bind vAppLib_hdl as super procedure' VIEW-AS ALERT-BOX.

/* Setup readobj to Query for in use tables */
RUN Initialize (INPUT "Order", INPUT-OUTPUT vOrder_hdl).
RUN Initialize (INPUT "Orderline", INPUT-OUTPUT vOrderline_hdl).
RUN OpenQuery IN vOrder_hdl (INPUT "").

```

Super Procedures, cont.

Super Procedures



ADD-SUPER-PROCEDURE Method

The ADD-SUPER-PROCEDURE Method binds AppLib1.w to w-order4.w.

Calculate Line Total

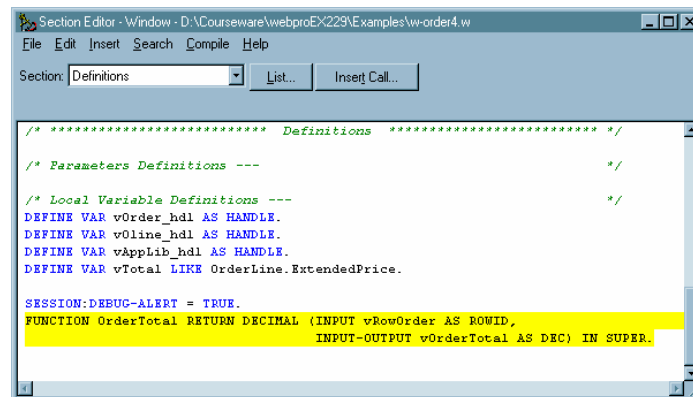
When the Calculate button is pressed, the above trigger runs LineTotal, which is an internal procedure residing in AppLib1.w (Super Procedure). Notice the absence of a handle indicating the location of LineTotal.

LineTotal

LineTotal calculates the ExtendedPrice based on the Qty, Price, and Discount.

Note The presence of AppLib2.w will be discussed in the context of the next example.

Super Procedures



```

Section Editor - Window - D:\Courseware\webproEX229\examples\vw-order4.w
File Edit Insert Search Compile Help
Section: Definitions List... Insert Cell...

/* ***** Definitions ***** */

/* Parameters Definitions --- */

/* Local Variable Definitions --- */
DEFINE VAR vOrder_hdl AS HANDLE.
DEFINE VAR vOrder_hdl AS HANDLE.
DEFINE VAR vAppLib_hdl AS HANDLE.
DEFINE VAR vTotal LIKE OrderLine.ExtendedPrice.

SESSION:DEBUG-ALERT = TRUE.
FUNCTION OrderTotal RETURN DECIMAL (INPUT vRowOrder AS ROWID,
                                     INPUT-OUTPUT vOrderTotal AS DEC) IN SUPER.

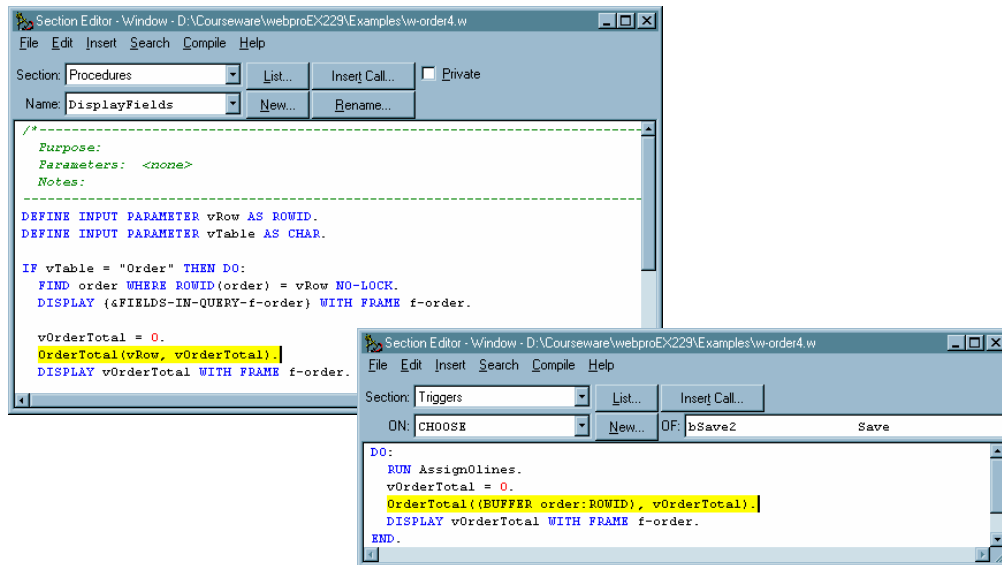
```

UDF Declaration

OrderTotal is a user-defined function defined in the super procedure AppLib1.w. Its purpose is to total the order lines for an order. To make a user-defined function accessible it must be forward declared using the SUPER Option on the forward declaration statement. This option tells the compiler the function exists in a Super Procedure bound later in the program.

Note The super procedure must be bound before accessing the function (ADD-SUPER-PROCEDURE).

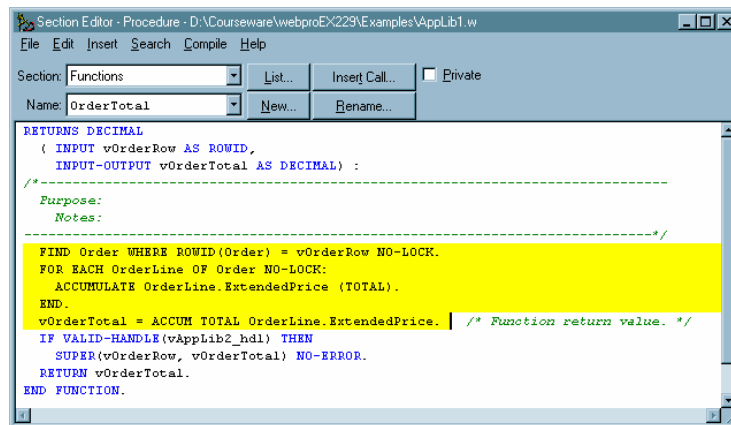
Super Procedures



UDF Usage

Each time a new order is read or new information in an order line is saved, the order line total is recalculated.

Super Procedures



```

Section Editor - Procedure - D:\Courseware\webproEX229\Examples\ApplLib1.w
File Edit Insert Search Compile Help
Section: Functions List... Insert Call... Private
Name: OrderTotal New... Rename...

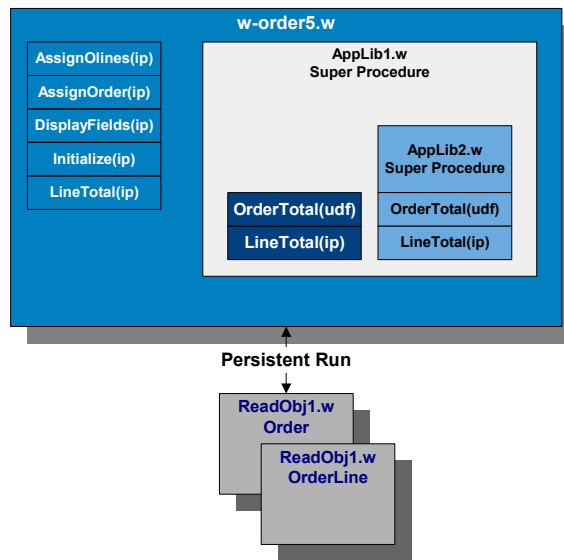
RETURNS DECIMAL
( INPUT vOrderRow AS ROWID,
  INPUT-OUTPUT vOrderTotal AS DECIMAL ) :
/*
Purpose:
Notes:
-----*/
FIND Order WHERE ROWID(Order) = vOrderRow NO-LOCK.
FOR EACH OrderLine OF Order NO-LOCK:
  ACCUMULATE OrderLine.ExtendedPrice (TOTAL).
END.
vOrderTotal = ACCUM TOTAL OrderLine.ExtendedPrice. /* Function return value. */
IF VALID-HANDLE(vApplLib2_hdl) THEN
  SUPER(vOrderRow, vOrderTotal) NO-ERROR.
RETURN vOrderTotal.
END FUNCTION.

```

UDF Code

Based on the supplied ROWID, the OrderTotal function calculates the order total, passing back the value in an INPUT-OUTPUT parameter.

Super Procedures



Procedure Calls

w-order5.w contains a local version of LineTotal. Like the w-order4.w it also set up AppLib1.w to be a Super Procedure. AppLib1.w also sets up a Super Procedure AppLib2.w containing a version of LineTotal. This set of examples contains three versions of LineTotal:

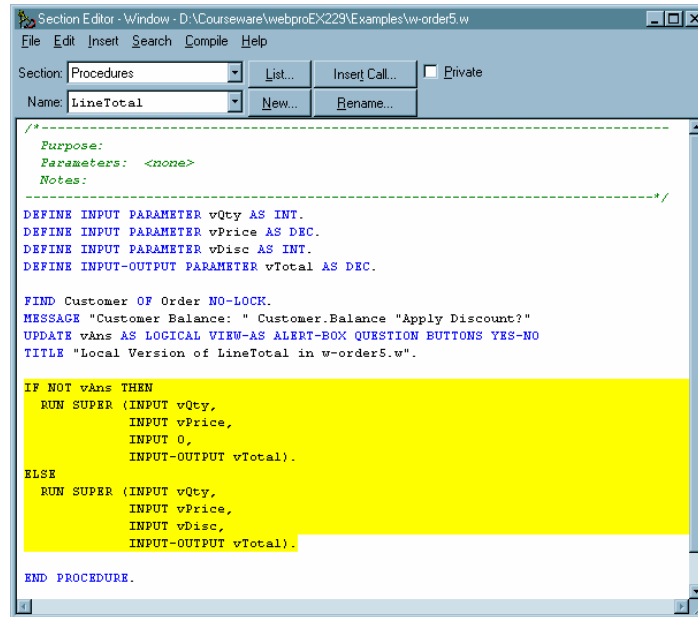
- LineTotal in w-order5.w
- LineTotal in AppLib1.w
- LineTotal in AppLib2.w

Search Order

When multiple versions of an internal procedure or user-defined function are present, Progress searches using the following order of precedence:

- 1** Search for a local internal procedure
- 2** Search for internal procedure in super procedure bound to current procedure
- 3** Search for an internal procedure in super procedure bound to SESSION
- 4** Search for external procedure

Super Procedures



```

Section Editor - Window - D:\Courseware\webproEX229\Examples\w-order5.w
File Edit Insert Search Compile Help
Section: Procedures List... Insert Call... Private
Name: LineTotal New... Rename...

/*
Purpose:
Parameters: <none>
Notes:
-----*/

DEFINE INPUT PARAMETER vQty AS INT.
DEFINE INPUT PARAMETER vPrice AS DEC.
DEFINE INPUT PARAMETER vDisc AS INT.
DEFINE INPUT-OUTPUT PARAMETER vTotal AS DEC.

FIND Customer OF Order NO-LOCK.
MESSAGE "Customer Balance: " Customer.Balance "Apply Discount?"
UPDATE vAns AS LOGICAL VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO
TITLE "Local Version of LineTotal in w-order5.w".

IF NOT vAns THEN
    RUN SUPER (INPUT vQty,
               INPUT vPrice,
               INPUT 0,
               INPUT-OUTPUT vTotal).
ELSE
    RUN SUPER (INPUT vQty,
               INPUT vPrice,
               INPUT vDisc,
               INPUT-OUTPUT vTotal).

END PROCEDURE.

```

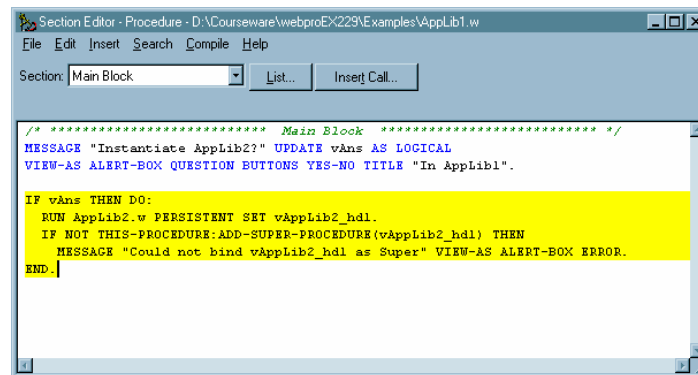
RUN SUPER Statement

Once a version of a procedure or function is found, it alone executes. However, a developer may wish to run the super version in addition to the current version of a procedure. To run the super version, use RUN SUPER Statement passing the appropriate parameters. In this way, a developer may omit or surround standard functionality found in a super procedure.

Syntax

```
RUN SUPER (input output parameter)
```

Super Procedures



```

Section Editor - Procedure - D:\Courseware\webproEX229\Examples\AppLib1.w
File Edit Insert Search Compile Help
Section: Main Block List... Insert Call...

/* ***** Main Block ***** */
MESSAGE "Instantiate AppLib2?" UPDATE vAns AS LOGICAL
VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO TITLE "In AppLib1".

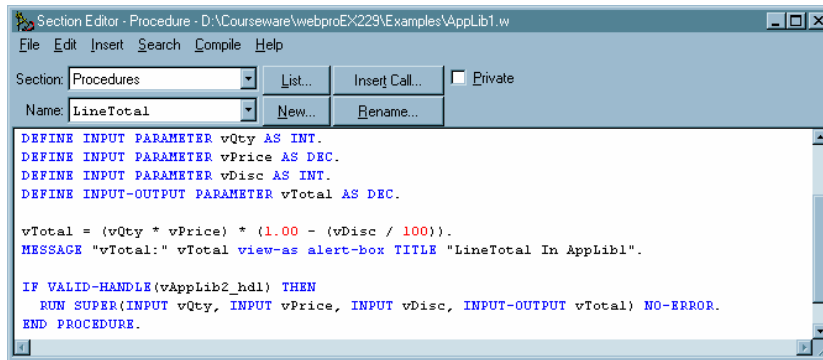
IF vAns THEN DO:
  RUN AppLib2.w PERSISTENT SET vAppLib2_hdl.
  IF NOT THIS-PROCEDURE:ADD-SUPER-PROCEDURE(vAppLib2_hdl) THEN
    MESSAGE "Could not bind vAppLib2_hdl as Super" VIEW-AS ALERT-BOX ERROR.
END.

```

Stacked Super Procedures

Another method for enhancing standard functionality is to bind an additional super procedure to an existing super procedure. AppLib2.w is bound to AppLib1.w as a super procedure. It contains yet another version of LineTotal, which provides some additional functionality.

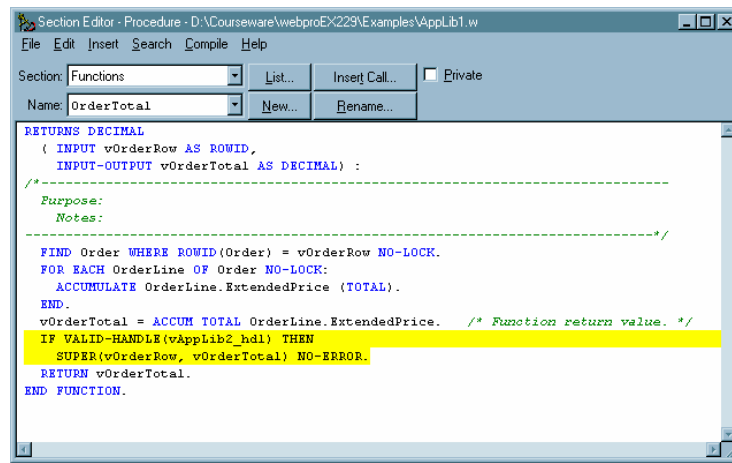
Super Procedures



Business Objects

Each function and internal procedure contained in AppLib1.w has a co-existing version in AppLib2.w. Each version in AppLib2.w is meant to enhance the standard functionality found in AppLib1.w. In this way, standard logic can be enhanced without physically changing the logic.

Super Procedures



```

Section Editor - Procedure - D:\Courseware\webproEX229\examples\Applib1.w
File Edit Insert Search Compile Help
Section: Functions List... Insert Call... Private
Name: OrderTotal New... Rename...

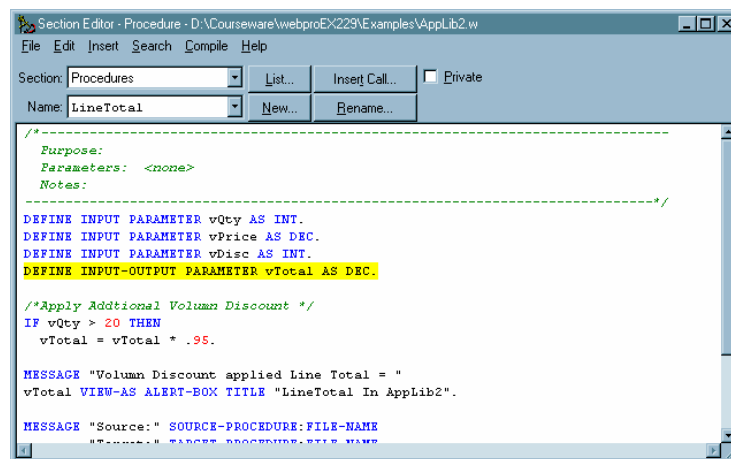
RETURNS DECIMAL
( INPUT vOrderRow AS ROWID,
  INPUT-OUTPUT vOrderTotal AS DECIMAL ) :
/*-----*/
Purpose:
Notes:
/*-----*/
FIND Order WHERE ROWID(Order) = vOrderRow NO-LOCK.
FOR EACH OrderLine OF Order NO-LOCK:
  ACCUMULATE OrderLine.ExtendedPrice (TOTAL).
END.
vOrderTotal = ACCUM TOTAL OrderLine.ExtendedPrice. /* Function return value. */
IF VALID-HANDLE(vApplib2_hdl) THEN
  SUPER(vOrderRow, vOrderTotal) NO-ERROR.
RETURN vOrderTotal.
END FUNCTION.

```

NO-ERROR Option

The NO-ERROR option is present on the SUPER Statement to suppress any error messages resulting from attempting to execute a non-existent version in AppLib2.w. You can build AppLib1.w based on the assumption that additional logic may be present in AppLib2.w, or added later.

Super Procedures



```

Section Editor - Procedure - D:\Courseware\webproEX229\Examples\AppLib2.w
File Edit Insert Search Compile Help
Section: Procedures List... Insert Call... Private
Name: LineTotal New... Rename...

/*-----
Purpose:
Parameters: <none>
Notes:
-----*/

DEFINE INPUT PARAMETER vQty AS INT.
DEFINE INPUT PARAMETER vPrice AS DEC.
DEFINE INPUT PARAMETER vDisc AS INT.
DEFINE INPUT-OUTPUT PARAMETER vTotal AS DEC.

/*Apply Additional Volume Discount */
IF vQty > 20 THEN
    vTotal = vTotal * .95.

MESSAGE "Volume Discount applied Line Total = "
vTotal VIEW-AS ALERT-BOX TITLE "LineTotal In AppLib2".

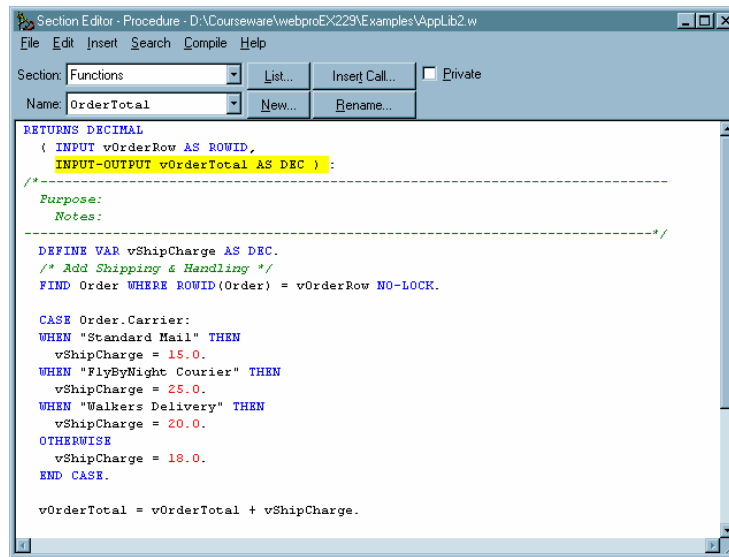
MESSAGE "Source: " SOURCE-PROCEDURE-FILE-NAME
"-----" TARGET-PROCEDURE-FILE-NAME

```

More Business Objects

To give the versions in AppLib2.w the ability to enhance the functionality in AppLib1.w, procedure and function results are returned using INPUT-OUTPUT parameters. That way, standard functionality can be applied in AppLib1.w and the result can be passed to the version in AppLib2.w for further manipulation.

Super Procedures



```

Section Editor - Procedure - D:\Courseware\webprod\EX229\Examples\Applib2.w
File Edit Insert Search Compile Help

Section: Functions List... Insert Call... Private
Name: OrderTotal New... Rename...

RETURNS DECIMAL
( INPUT vOrderRow AS ROWID,
  INPUT-OUTPUT vOrderTotal AS DEC ):
/*-----
Purpose:
Notes:
-----*/

DEFINE VAR vShipCharge AS DEC.
/* Add Shipping & Handling */
FIND Order WHERE ROWID (Order) = vOrderRow NO-LOCK.

CASE Order.Carrier:
WHEN "Standard Mail" THEN
  vShipCharge = 15.0.
WHEN "FlyByNight Courier" THEN
  vShipCharge = 25.0.
WHEN "Walkers Delivery" THEN
  vShipCharge = 20.0.
OTHERWISE
  vShipCharge = 18.0.
END CASE.

vOrderTotal = vOrderTotal + vShipCharge.

```

Super Procedures, cont.

Dynamic Objects

Introduction

- Dynamic Queries, Buffers and Browsers provide a developer the ability to bring an application component into existence at run-time. Developers may also conditionally change the sort criteria or record reading constraints at run time.

Benefits

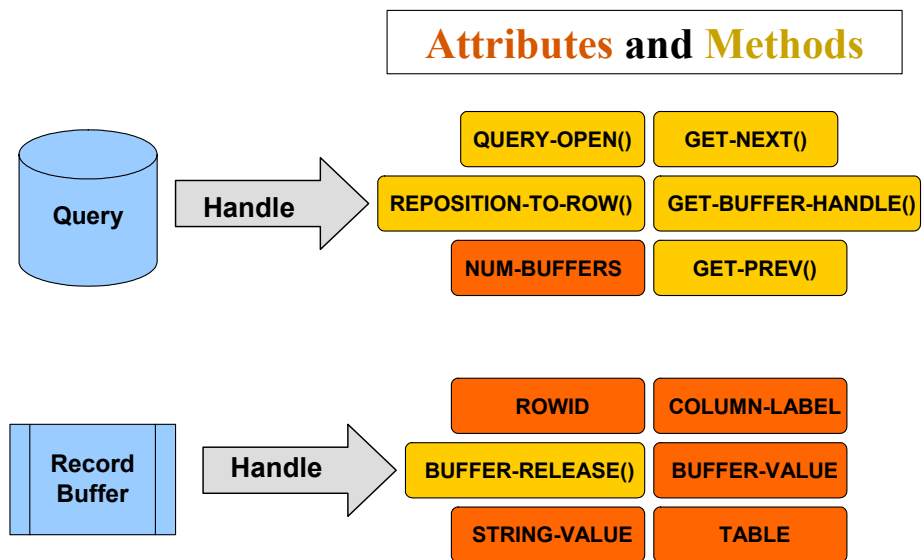
- Record reading constraints and sort criteria are easily manipulated at run-time.
- The ability to dynamically define Queries and Buffers allows a developer to create re-usable application components.
- Since a Query can instantiate itself at run-time, its constraints can be determined by the value/mode of certain UI objects. In effect a Dynamic Query can maintain itself which leads to a more easily maintained application.

Dynamic Objects

Introduction

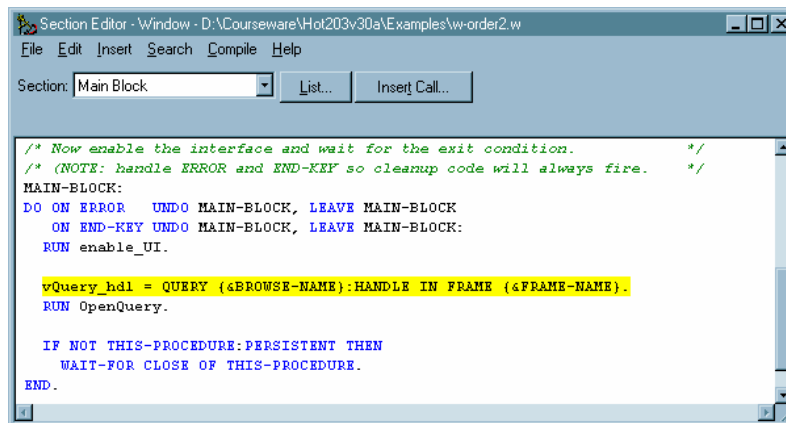
In the CSS product, you will find both dynamic queries and buffers. To fully understand CSS you will need to understand how they work.

Dynamic Objects



Attributes and Methods

Dynamic Queries



```

Section Editor - Window - D:\Courseware\Hot203v30a\Examples\w-order2.w
File Edit Insert Search Compile Help
Section: Main Block List... Insert Call...

/* Now enable the interface and wait for the exit condition. */
/* (NOTE: handle ERROR and END-KEY so cleanup code will always fire. */
MAIN-BLOCK:
DO ON ERROR UNDO MAIN-BLOCK, LEAVE MAIN-BLOCK
ON END-KEY UNDO MAIN-BLOCK, LEAVE MAIN-BLOCK:
RUN enable_UI.

vQuery_hdl = QUERY (&BROWSE-NAME):HANDLE IN FRAME (&FRAME-NAME).
RUN OpenQuery.

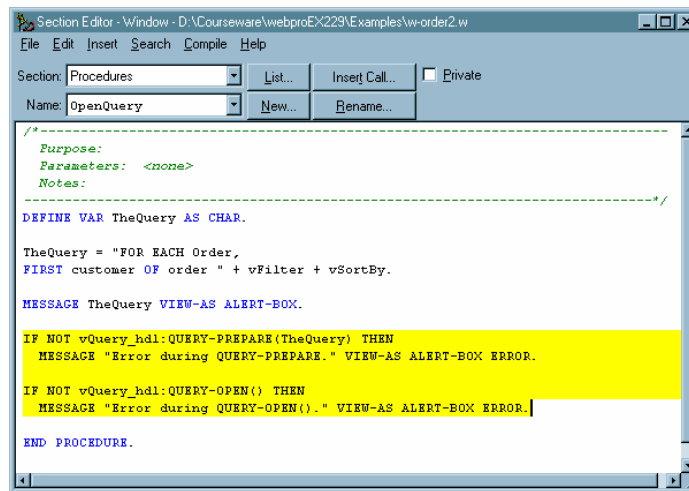
IF NOT THIS-PROCEDURE:PERSISTENT THEN
WAIT-FOR CLOSE OF THIS-PROCEDURE.
END.

```

Query Handles

The first step in dynamically manipulating a query is to reference its handle. A query's handle is an attribute that is only available in V9. Once obtained, it provides access to a variety of new attributes and methods related to a query. In this example, the handle of the order browser's query is stored for later use.

Dynamic Queries



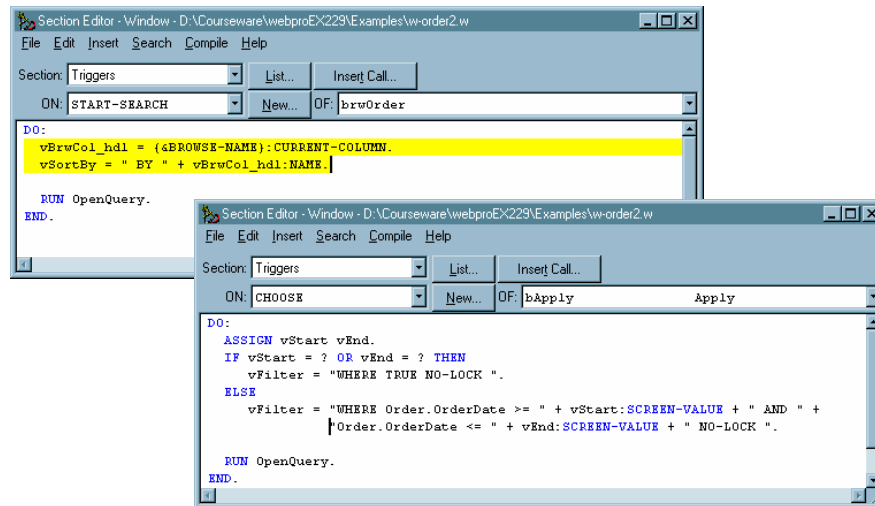
QUERY-PREPARE() Method

When the QUERY-PREPARE Method is encountered at run-time, it compiles a predicate into a query. The predicate portion of a query supplies tables, record reading constraints, and sort criteria.

QUERY-OPEN() Method

This method opens the referenced query. You must successfully execute a QUERY-PREPARE() before executing QUERY-OPEN(). You may execute QUERY-OPEN() as many times as desired for a given query without re-executing QUERY-PREPARE(). QUERY-PREPARE() only needs to be re-executed when the nature of the query changes.

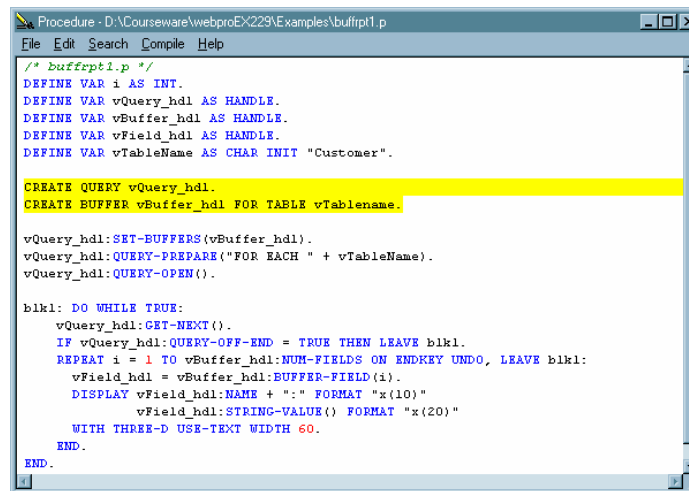
Dynamic Queries



Query Settings

Two variables (vSortBy and vFilter) maintain Sort and Filter settings. Each variable is used in the construction of the query string in OpenQuery.

Dynamic Queries & Buffers



```

/* buffpt1.p */
DEFINE VAR i AS INT.
DEFINE VAR vQuery_hdl AS HANDLE.
DEFINE VAR vBuffer_hdl AS HANDLE.
DEFINE VAR vField_hdl AS HANDLE.
DEFINE VAR vTableName AS CHAR INIT "Customer".

CREATE QUERY vQuery_hdl.
CREATE BUFFER vBuffer_hdl FOR TABLE vTableName.

vQuery_hdl:SET-BUFFERS(vBuffer_hdl).
vQuery_hdl:QUERY-PREPARE("FOR EACH " + vTableName).
vQuery_hdl:QUERY-OPEN().

blk1: DO WHILE TRUE:
  vQuery_hdl:GET-NEXT().
  IF vQuery_hdl:QUERY-OFF-END = TRUE THEN LEAVE blk1.
  REPEAT i = 1 TO vBuffer_hdl:NUM-FIELDS ON ENDKEY UNDO, LEAVE blk1:
    vField_hdl = vBuffer_hdl:BUFFER-FIELD(i).
    DISPLAY vField_hdl:NAME + ":" FORMAT "x(10)"
    vField_hdl:STRING-VALUE() FORMAT "x(20)"
  WITH THREE-D USE-TEXT WIDTH 60.
  END.
END.

```

Dynamic Queries and Buffers

CREATE Widget Statement

The CREATE Statement can be used to bring queries and buffers into existence at run-time.

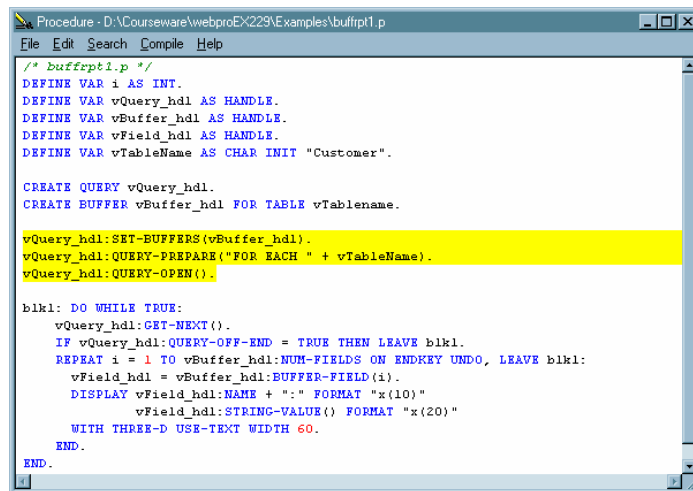
Syntax

CREATE QUERY supplied-handle.

CREATE BUFFER supplied-handle FOR "table-name"

Note The "table-name" may be a hard-coded string or supplied via a variable

Dynamic Queries & Buffers



```

Procedure - D:\Courseware\webproEX229\Examples\buffrpt1.p
File Edit Search Compile Help

/* buffrpt1.p */
DEFINE VAR i AS INT.
DEFINE VAR vQuery_hdl AS HANDLE.
DEFINE VAR vBuffer_hdl AS HANDLE.
DEFINE VAR vField_hdl AS HANDLE.
DEFINE VAR vTableName AS CHAR INIT "Customer".

CREATE QUERY vQuery_hdl.
CREATE BUFFER vBuffer_hdl FOR TABLE vTableName.

vQuery_hdl:SET-BUFFERS(vBuffer_hdl).
vQuery_hdl:QUERY-PREPARE("FOR EACH " + vTableName).
vQuery_hdl:QUERY-OPEN().

bkl1: DO WHILE TRUE:
  vQuery_hdl:GET-NEXT().
  IF vQuery_hdl:QUERY-OFF-END = TRUE THEN LEAVE bkl1.
  REPEAT i = 1 TO vBuffer_hdl:NUM-FIELDS ON ENDKEY UNDO, LEAVE bkl1:
    vField_hdl = vBuffer_hdl:BUFFER-FIELD(i).
    DISPLAY vField_hdl:NAME + ":" FORMAT "x(10)"
    vField_hdl:STRING-VALUE() FORMAT "x(20)"
  WITH THREE-D USE-TEXT WIDTH 60.
  END.
END.

```

Buffer Handle

HANDLE is a new attribute for buffers in V9. Once a handle is obtained for a buffer, there are numerous attributes and methods, which can be used to access and manipulate a buffer during program execution.

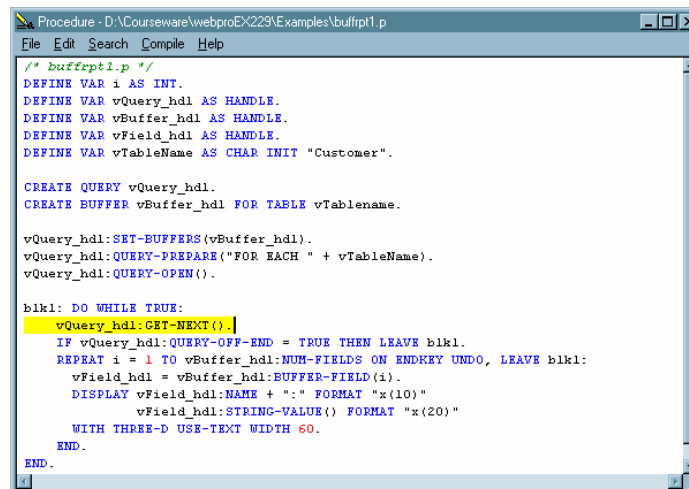
FIND FIRST customer.

MESSAGE BUFFER customer:HANDLE VIEW-AS ALERT-BOX.

SET-BUFFERS () Method

This method binds a buffer to a query at run-time. Similar to a “DEFINE QUERY” statement used at compile time.

Dynamic Query Manipulation



```

Procedure - D:\Courseware\webproEX229\Examples\buffrpt1.p
File Edit Search Compile Help

/* buffrpt1.p */
DEFINE VAR i AS INT.
DEFINE VAR vQuery_hdl AS HANDLE.
DEFINE VAR vBuffer_hdl AS HANDLE.
DEFINE VAR vField_hdl AS HANDLE.
DEFINE VAR vTableName AS CHAR INIT "Customer".

CREATE QUERY vQuery_hdl.
CREATE BUFFER vBuffer_hdl FOR TABLE vTableName.

vQuery_hdl:SET-BUFFERS(vBuffer_hdl).
vQuery_hdl:QUERY-PREPARE("FOR EACH " + vTableName).
vQuery_hdl:QUERY-OPEN().

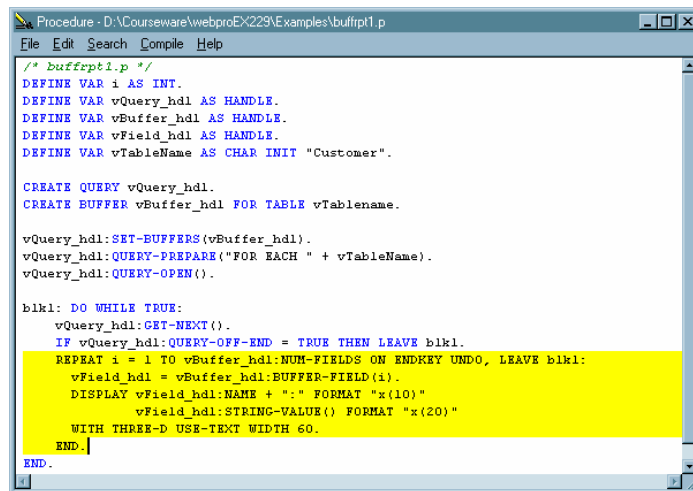
bkl1: DO WHILE TRUE:
    vQuery_hdl:GET-NEXT().
    IF vQuery_hdl:QUERY-OFF-END = TRUE THEN LEAVE bkl1.
    REPEAT i = 1 TO vBuffer_hdl:NUM-FIELDS ON ENDKEY UNDO, LEAVE bkl1:
        vField_hdl = vBuffer_hdl:BUFFER-FIELD(i).
        DISPLAY vField_hdl:NAME + ":" FORMAT "x(10)"
        vField_hdl:STRING-VALUE() FORMAT "x(20)"
        WITH THREE-D USE-TEXT WIDTH 60.
    END.
END.
  
```

Dynamic Query Manipulation

GET-FIRST(), GET-LAST(), GET-PREV(), GET-NEXT() Methods

All of these methods are used to retrieve the desired record from a set of records defined by a query. Remember that the GET NEXT, PREV, FIRST, LAST statements only work with a query defined at compile time (static). A Method can work with a query's handle regardless of when the query was brought into existence.

Dynamic Buffer Fields



```

Procedure - D:\Courseware\webproEX229\Examples\buffrpt1.p
File Edit Search Compile Help
/* buffrpt1.p */
DEFINE VAR i AS INT.
DEFINE VAR vQuery_hdl AS HANDLE.
DEFINE VAR vBuffer_hdl AS HANDLE.
DEFINE VAR vField_hdl AS HANDLE.
DEFINE VAR vTableName AS CHAR INIT "Customer".

CREATE QUERY vQuery_hdl.
CREATE BUFFER vBuffer_hdl FOR TABLE vTableName.

vQuery_hdl:SET-BUFFERS(vBuffer_hdl).
vQuery_hdl:QUERY-PREPARE("FOR EACH " + vTableName).
vQuery_hdl:QUERY-OPEN().

bkl1: DO WHILE TRUE:
    vQuery_hdl:GET-NEXT().
    IF vQuery_hdl:QUERY-OFF-END = TRUE THEN LEAVE bkl1.
    REPEAT i = 1 TO vBuffer_hdl:NUM-FIELDS ON ENDKEY UNDO, LEAVE bkl1:
        vField_hdl = vBuffer_hdl:BUFFER-FIELD(i).
        DISPLAY vField_hdl:NAME + ":" FORMAT "x(10)"
        vField_hdl:STRING-VALUE() FORMAT "x(20)"
        WITH THREE-D USE-TEXT WIDTH 60.
    END.
END.

```

Dynamic Buffer Fields

BUFFER-FIELD() Method

In addition to the attributes and methods of a buffer, there are also numerous attributes and methods associated with a particular field in a buffer. Access to a field (buffer field object) can only be gained via the field's handle. The BUFFER-FIELD() Method returns the handle of the requested field. Once the handle of the field is obtained a developer may then manipulate a field in a buffer.

Syntax

vFieldHandle = vBufferHandle:BUFFER-FIELD(n)

Note Where “n” is an integer that represents the ordinal position held by a field in a table. This is the field order number stored with each field as it is defined.

NAME Attribute & STRING-VALUE() Method

The NAME Attribute and STRING-VALUE Method are used to access the name and value of the field identified by the previously executed BUFFER-FIELD() Method.

Publish and Subscribe

▲ Introduction

- Also referred to as Named Events
- Supported in Progress version 9+
- Invoked and managed using the PUBLISH and SUBSCRIBE 4GL statements

Publish and Subscribe

Introduction

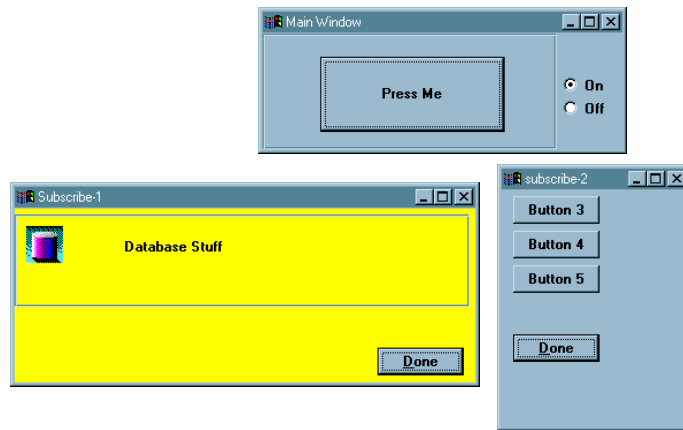
Publish and Subscribe

▲ Benefits

- Reusable code
- Encapsulation of functionality
- Eliminates the need to keep up with persistent procedure handles.

Benefits

Publish and Subscribe



Named Events

A Named Event can be set up or broadcast using the PUBLISH Statement. A developer programmatically causes a Named Event to occur by executing a PUBLISH Statement. Similar behavior occurs when an APPLY Statement programmatically causes a User Interface Event to occur.

Other procedures may define a coded response to the published event by issuing the SUBSCRIBE Statement. Notice the subscribing procedures define an Internal Procedure whose name matches the name of the published event. This procedure contains the coded response to the published event.

Dynamic HTML

- ▲ What is DHTML?
 - Standard HTML is a combination of different elements, or tags, which are designed to hold data in a certain fashion
 - Dynamic HTML extends these tags by making them “objects”, and allowing you to alter the way the data is presented by altering those tag “objects”
 - Examples of DHTML objects include <body>, <title>, <src>, <input>, bgcolor, forecolor, value, etc.

Dynamic HTML

Dynamic HTML

- ▲ Examples of DHTML in CSS:
 - Creation of menu bars
 - Changing values and validating form fields
 - Layers
 - Performing document.write

Dynamic HTML, cont.

Lab Exercises: Technologies Applied

- ▲ Perform Lab activities for Chapter 5
- ▲ They are found in the Labs Appendix

Lab Exercises

CHAPTER 6

CSS Architecture

Naming Conventions

- ▲ Directories and File Names
 - Lower case characters only
 - 2 – 3 character module directory names
 - All the file names will start with the module directory prefix
 - Use only “_” (underscores) in directory and file names and not “-” (hyphens) for separation.

Naming Conventions

Name Format

Module_dir/mod_func_unique-name(n).ext

<i>Module_dir:</i>	Module directory name
<i>Mod:</i>	Module reference
<i>Func:</i>	Type of procedure (see tables below)
<i>Unique-name:</i>	Verbose filename descriptor
<i>N:</i>	Sequence name of procedure if applicable
<i>Ext:</i>	Extension (html, i, p)

Func	Description
Adm	Administration
B2c	Business to customer
Brw	Browse
Cc	Credit card
Edt	Maintenance edit
Eml	Email
Ext	Extension

Func	Description
Inq	Inquiry
Mnt	Maintenance nav
Op	Order processing
Rp	Report
Rpe	Report body entry
Rpt	Report body
Sys	System

Naming Conventions

▲ Variables

- Leading character in variable names indicate the data-type.
- Hungarian Notation used in creation of variable names.

▲ Temp-Tables

- Names begin with prefix "tt_" followed by the physical table name (i.e., tt_wpro_cust_num).
- Temp-tables not defined like database tables will have an additional component after the "tt_" prefix. (i.e., tt_rpp_customtable).

Variable Name Codes

Variable names will begin with one of the following codes in lowercase:

c	Decimal	h	Handle
i	Integer	r	Rowid
d	Date	rd	Recid
k	Character	rw	Raw
l	Logical		

For further understanding on Hungarian notation, please refer to the following link:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvsngen/html/hunganotat.asp>

Temp-Table Standards

All temp-tables will be defined, at a minimum, as defined below. They must contain NO-UNDO.

```
DEFINE TEMP-TABLE tt_wpro_cust_item NO-UNDO LIKE wpro_cust_item
FIELD tbSrtOrder AS INTEGER
FIELD tbROWID AS ROWID.
```

Naming Conventions

▲ Internal Procedures and Functions

Procedure and Function names will begin with a lowercase prefix from the following list:

Prefix	Indicates the procedure or function will...
get	return a value
set	update a db record
put	output some HTML
val	apply an EX or UI business rule
gen	generate a screen or screen section
wrap	convert a procedure to a function

Internal Procedures and Functions

Naming Conventions

▲ Comments

- Procedures and functions must contain the following comment body:

```
Function Comment:
/** CSS Documentation ****
  Procedure/Function Name:
    Purpose:
  Return Value:
  Parameters:
    Input:
    Output:
  Input/Output:
**** End CSS Documentation ***/

{&EXDEBUG}
```

Comments

Procedure/Function Name:	Enter the procedure or function name as defined. All functions should include the following parenthesis.
Purpose:	If the purpose is known or specified, it should be a short precise explanation of the contained behavior.
Return Value:	Specify the defined return value for all functions. Insert NONE for procedures.
Parameters Input:	Specify each parameter on a separate line. Each one must specify its data type.
Output:	Same as above. If no output parameters exist, specify "none."
Input/Output:	Same as above.

Note Every internal procedure and user-defined function will contain the {&EXDEBUG} preprocessor directive inserted alone on a line following all input/output parameters and variable definitions.

Lab Exercise – Naming Conventions

- ▲ Perform Lab Activities for Chapter 6. They are found in the Labs Appendix.

Lab Exercise

Data Architecture

- ▲ CSS Database Schema
- ▲ Over 85 tables
- ▲ Complete list in CSS Schema Appendix
- ▲ Key Tables:

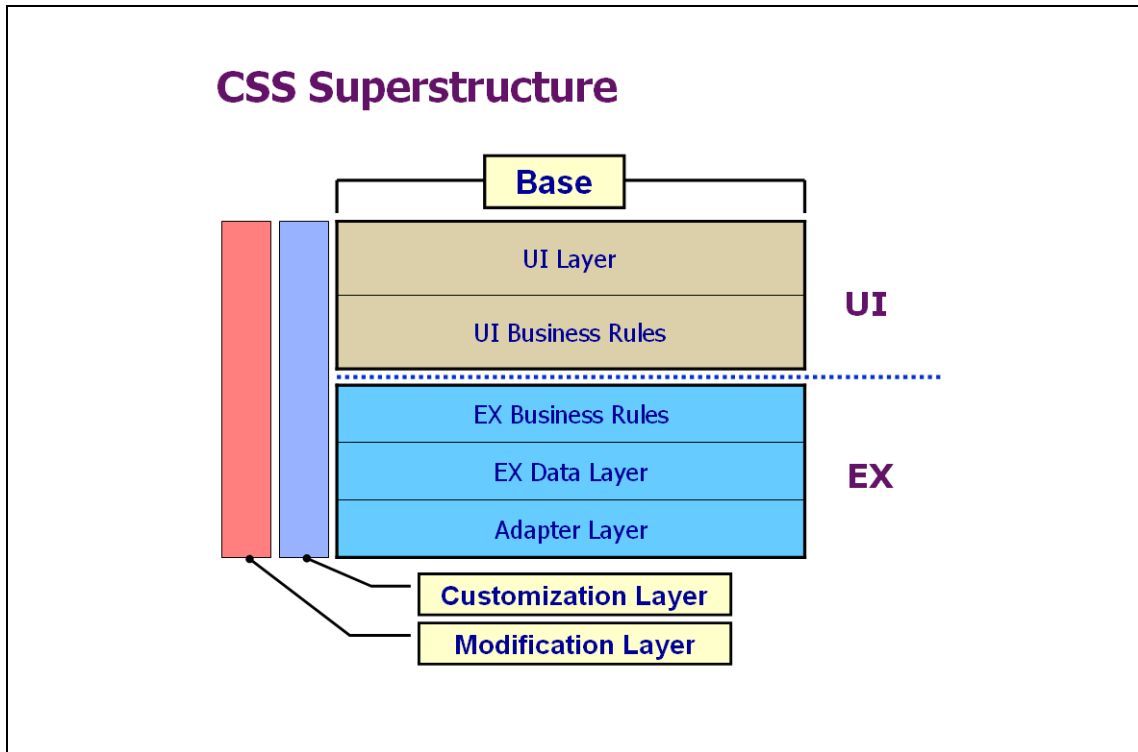
wpro_browse_mstr	wpro_lang_control	wpro_Registry
wpro_cust	wpro_menu	wpro_session
wpro_cust_item	wpro_menuButtons	wpro_string_trans
wpro_data_trans	wpro_module	wpro_super
wpro_debug	wpro_module_js	wpro_user_detail
wpro_item_types	wpro_order_control	wpro_user_master

Data Architecture

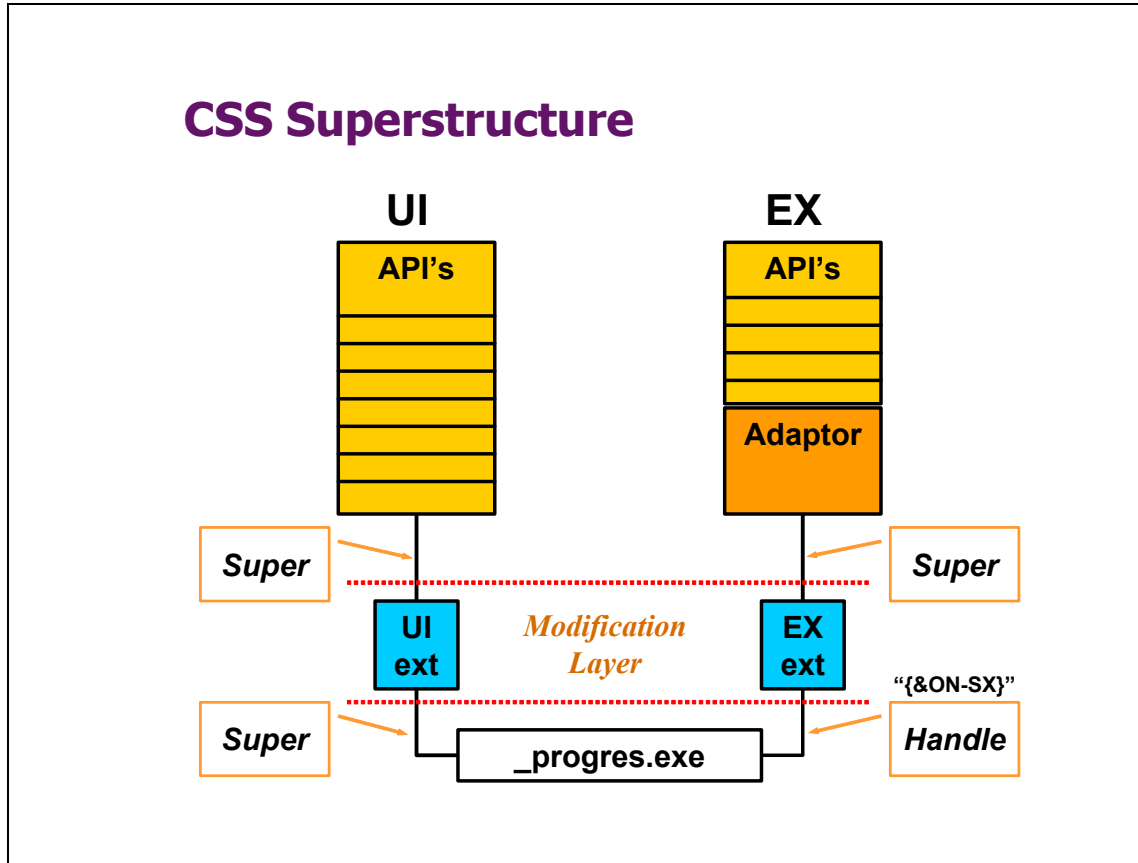
CSS Superstructure

- ▲ UI Layers
- ▲ EX Layers
- ▲ Adapter Layer
- ▲ Modification Layer

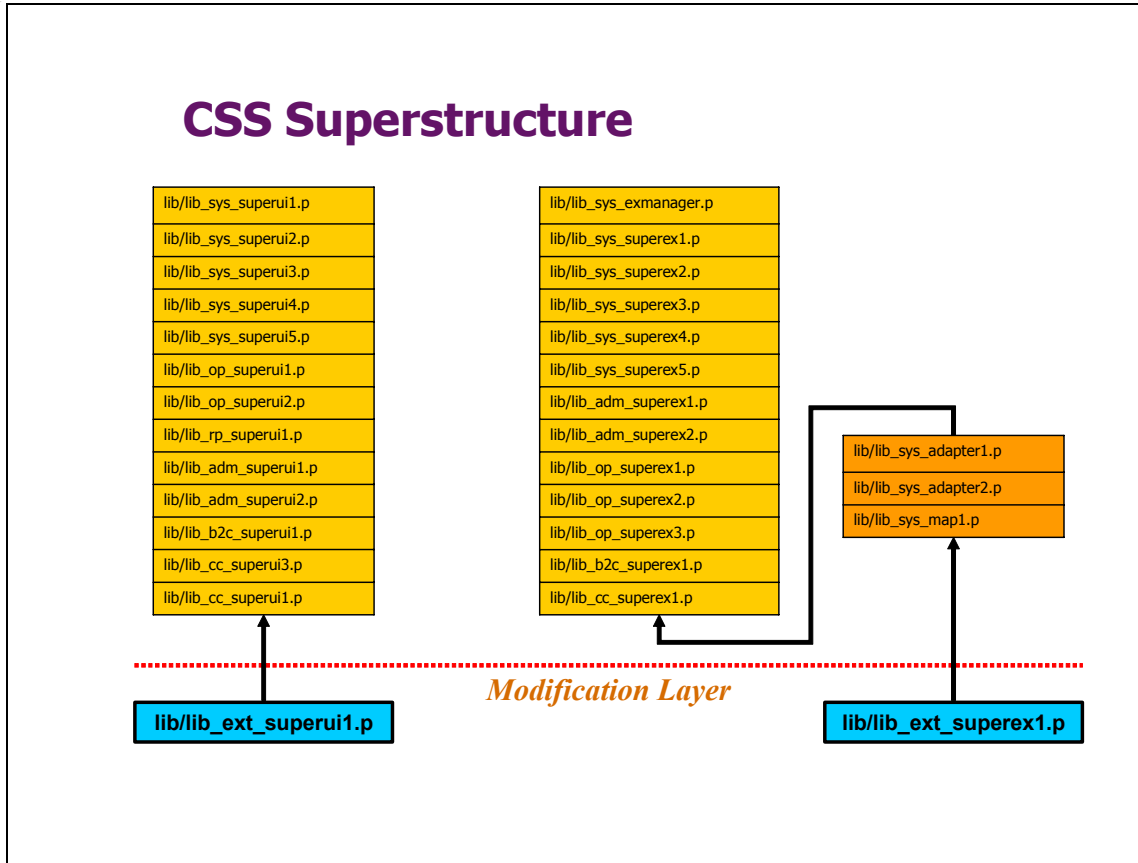
CSS Superstructure



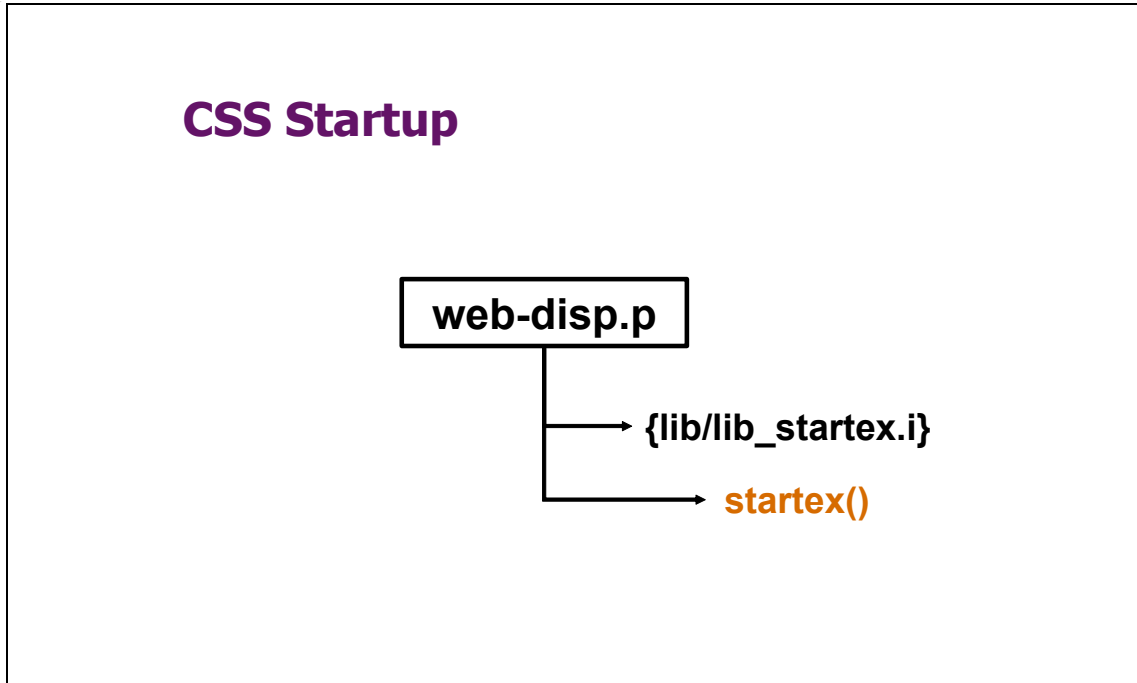
CSS Superstructure, cont.



CSS Superstructure, cont.

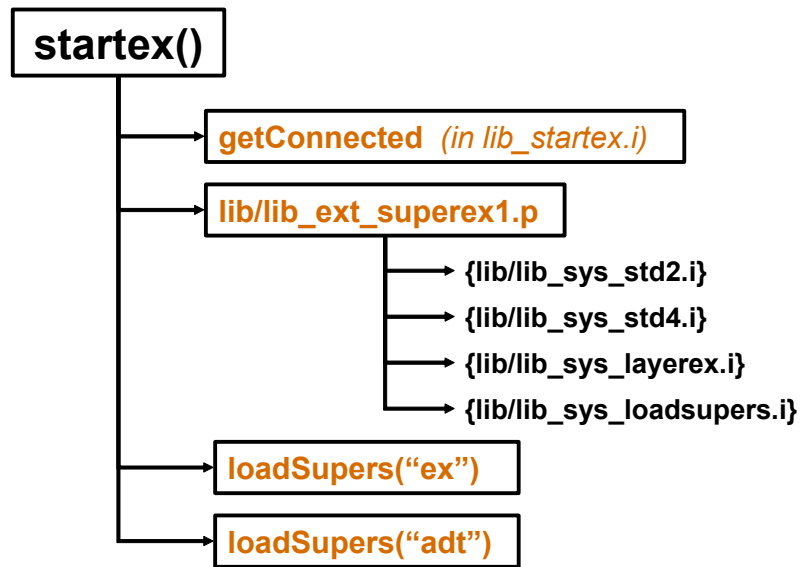


CSS Superstructure, cont.



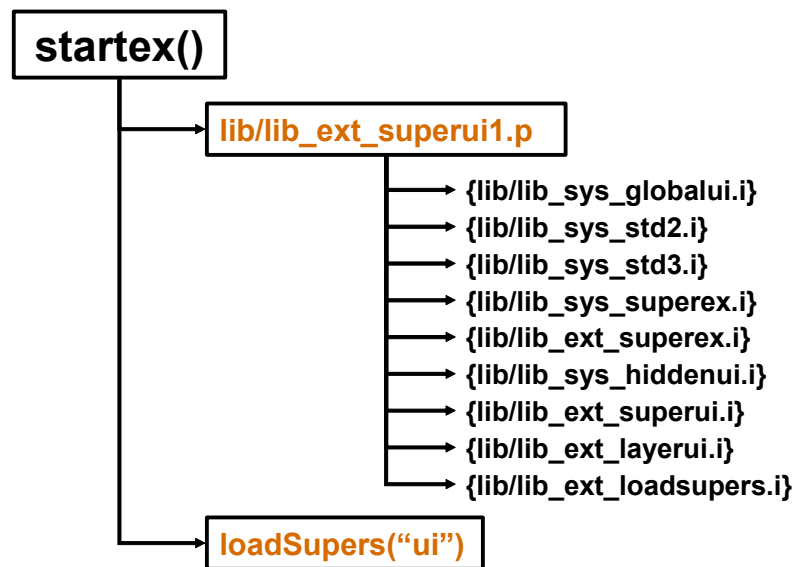
CSS Startup

CSS Startup



CSS Startup, cont.

CSS Startup



CSS Startup, cont.

CSS Startup

LoadSupers:

```
RUN loadSupers IN hwsx (INPUT expV, "Extended", ...  
RUN loadSupers IN hwsx (INPUT expV, "Adapter", ...  
RUN loadSupers IN hwui (INPUT expV, "UI", ...
```

CSS Startup, cont.

CHAPTER 7

Customization

Customization

▲ Overview

- Basic template
- Maintenance template
- Report template
- Key API's
- Utilizing the super procedures (creating a new layer)
- Customizing your procedures
- Modifying the core product

Customization

Overview

Basic Template

- ▲ UI container
- ▲ All other templates start from this template
- ▲ Contains the base CSS objects
- ▲ Defines five basic regions

Basic Template

Basic Template

▲ Overview of components:

- lib/lib_std.i
- {tt_tablename.i}
- putEXheader
- insertNewHeader
- genLeft
- genCenter
- genRight
- insertNewFooter

Components

The basic template is comprised of eight components:

- 1 lib/lib_std.i: manages session state
- 2 {tt_tablename.i}: temp-table definition
- 3 putEXheader: outputs the html header
- 4 insertNewHeader: outputs the CSS navigation bar
- 5 genLeft: outputs html content for left panel
- 6 genCenter: outputs html content for center panel
- 7 genRight: outputs html content for right panel
- 8 insertNewFooter: outputs the CSS standard html footer

Basic Template

- ▲ lib/lib_std.i
 - Includes lib/lib_sys_std1.i
 - Defines four variables
 - userHomePage
 - and_so_on
 - vPageTitle
 - kDebug
 - Validates specific web values
 - screenHeight
 - browserName
 - browserVersion
 - Redirects to start page for initial requests
 - Executes checkSession()

lib/lib_std.i

This standard include is on every generated html page. It establishes and controls how CSS manages session and the web agents to all other CSS objects.

The include file lib/lib_sys_std1.i links together the other lib_sys_std#.i required for a UI object.

The userHomePage variable controls where the user is redirected when they press the “Home” button.

The and_so_on variable contains the url query string of required form variables needed when opening a CSS page through Javascript.

vPageTitle contains the window title value.

kDebug contains the state of the debug feature: either off or on.

The validated web values must be populated. When they are not, the user will be redirected to the start page. The CSS start page determines the user's browser settings and returns the requested page to the user.

The `checkSession()` function creates and updates the CSS system with the user's session information. This function also applies application security and url socket security.

Basic Template

▲ {tt_tablename.i}

- Temp-table definitions are stored in the tt directory.
- The include file has the same name of the temp-table defined within.
- The *tablename* string represents the physical database table name or unique UI form name.
- All temp-tables include two required fields.
 - tbsrtOrder
 - tbROWID

{tt_tablename.i}

Example

```
DEFINE TEMP-TABLE tt_wpro_class NO-UNDO LIKE wpro_class
  FIELD tbsrtOrder AS INTEGER
  FIELD tbROWID AS ROWID.
```

Basic Template

- ▲ putEXHeader
 - Inserts the <META> tag for the char-set
 - Inserts CSS links
 - Inserts shortcut icon
 - Inserts <META http-equiv> tag for page expiration

putEXHeader

Example

```
<meta content="text/html; charset=iso-8859-1" http-equiv="Content-Type">
<title>Login</title>
<LINK REL="stylesheet" TYPE="text/css" HREF="/class4/styles/gen_webpro.css">
<LINK REL="stylesheet" TYPE="text/css" HREF="/class4/styles/ie_webpro.css">
<LINK REL="SHORTCUT ICON" HREF="/class4/images/favicon.ico">
<META HTTP-EQUIV="Expires" CONTENT="Tue, 04 Dec 1993 21:29:02 GMT">
```

Basic Template

▲ insertNewHeader

- Called in the top panel of the template.
- Requires script/standard.js
- Outputs the basic CSS menubar.
- Menubar is maintained in Menu Button Maintenance.

insertNewHeader

Basic Template

- ▲ Panel Generation
(genLeft, genCenter, genRight)
 - Three default panels
 - Left panel typically contains navigational features.
 - Center panel must contain one form with the form name of "centerForm".
 - Right panel contains additional information and is optional

Panel Generation

Basic Template

- ▲ insertNewFooter
 - Outputs the standard CSS page footer
 - Called in the bottom panel of the template.
 - Requires it's own <TR> tags.

insertNewFooter

Basic Template

- ▲ Create a new program
 1. Copy base template to custom module directory and rename.
 2. Make modifications
 3. Setting up CSS
 - a) System Module Maintenance
 - b) Menu Maintenance
 4. Test your code

Create New Program

Note Required Javascript libraries: standard1.js

Case Study – Lab 1

- ▲ Do Lab 1 of Chapter 7 Lab Exercises found in Appendix A

Case Study, Lab 1

Basic Template

- ▲ Page navigation and posting
 - preSubmit
 - centerForm
 - Required Fields
 - method=Post
 - Recover – get-Value

Page Navigation and Posting

Case Study – Lab 2

- ▲ Do Lab 2 of Chapter 7 Lab Exercises found in Appendix A

Case Study, Lab 2

Maintenance Template

- ▲ Built from base template
- ▲ Two html programs required for a maintenance module (**mnt** and **edt**)
- ▲ In-line instructions
- ▲ Quick with override points

Maintenance Template

Overview

Maintenance Template (mnt)

- ▲ Additional features in mnt template
 - deleteRecord
 - getParentJoin
 - genLeftMaintenance
 - genCenterMaintenance
 - genRightMaintenance

Additional Features

The maintenance navigation template (mnt) introduces five additional components:

- 1 deleteRecord: deletes record from database
- 2 getParentJoin: allows for chaining of different maintenance programs
- 3 genLeftMaintenance: outputs html content for standard left maintenance panel
- 4 genCenterMaintenance: outputs html content for standard center maintenance panel
- 5 genRightMaintenance: outputs html content for standard right maintenance panel

Maintenance Template (mnt)

- ▲ deleteRecord
 - Deletes the selected record based on input ROWID
 - Cleans up any data translation records
 - Allows a common place for handling of parent / child records

deleteRecord

Example

```

Procedure/Function Name: deleteRecord
    Purpose: Delete special master records and all related
              child specials records.
Return Value:
Parameters:
    Input: kSession    AS CHARACTER
           kTableName AS CHARACTER
           tbROWID    AS CHARACTER
    Output: kgetTotal  AS CHARACTER
           cErrorMsg  AS CHARACTER
Input/Output: none
    
```

Maintenance Template (mnt)

▲ getParentJoin

- Adjust the query phrase
 - Find parent record based on passed parent rowid.
 - Constrains child records to matching parent value(s).
- Returns properly formatted query phrase modifier
- Allows maintenance programs to reflect parent / child relationships.

getParentJoin

Example

```
PROCEDURE getParentJoin:
  DEFINE INPUT PARAMETER ksession AS CHARACTER.
  DEFINE INPUT PARAMETER kkey AS CHARACTER.
  DEFINE OUTPUT PARAMETER kJoin AS CHARACTER.
  DEFINE VARIABLE kROWID AS CHARACTER NO-UNDO.
  DEFINE VARIABLE kCustNum AS CHARACTER NO-UNDO.

  {&EXDEBUG}

  /* STEP 17 If this is a child program you need to
  populate the joinphrase. */
  ASSIGN kROWID = getKeyvalue(INPUT ksession, INPUT kkey).
  IF kROWID <> "" THEN
  DO:
    FIND FIRST wpro_cust WHERE
      ROWID(wpro_cust) = TO-ROWID(kROWID)NO-LOCK NO-ERROR.
    kCustNum = wpro_cust.customer_number.
    IF AVAIL(wpro_cust) THEN
      ASSIGN kJoin = " andwpro_cust_item.customer_number " +
        " = '" + kCustNum + "'" .
  END.
END PROCEDURE.
```

Maintenance Template (mnt)

- ▲ genLeftMaintenance
 - Adds a link back to parent maintenance program

genLeftMaintenance

Maintenance Template (mnt)

- ▲ genCenterMaintenance
 - Outputs in html table format an overview of the database table.
 - Provides for navigation of selected table by:
 - key value search
 - column sort
 - page navigation
 - Allows for table results size selection.
 - Works in conjunction with maintenance template to maintain web state information.
 - Encapsulates several CSS objects into one API call.

genCenterMaintenance

Maintenance Template (mnt)

- ▲ genRightMaintenance
 - Inserts a blank panel by default.
 - Can be overridden to allow for custom use of the right panel.

genRightMaintenance

Maintenance Template (edt)

▲ New features in edt template

- putEditHeader
- putEditButtons
- setLocalRecord
 - getValidateRecord
 - copyDataTrans
- genEditCenter
 - putInputText
 - putTextArea
 - putCheckBox

New Features in Edit Template

The maintenance edit template (edt) introduces six additional components:

- 1 putEditHeader: outputs non-navigable page header
- 2 putEditButtons: outputs standard Add/Save/Cancel buttons
- 3 setLocal Record: responsible for saving the modified record to the database
- 4 getValidateRecord: customer area for data validation
- 5 copyDataTrans: keeps data translation records in sync
- 6 genEditCenter: outputs the main body content of the window

Maintenance Template (edt)

- ▲ putEditHeader
 - Replaces the standard menu bar with a function title bar.
 - Ensures popup window has recognizable title.

putEditHeader

Maintenance Template (edt)

- ▲ putEditButtons
 - Inserts standard edit buttons.
 - Includes Javascript to post correct form state.
 - Resides within its own table definition.

putEditButtons

Maintenance Template (edt)

- ▲ setLocalRecord
 - Provides common place for handling record update.
 - Calls local data validation routine getValidateRecord.
 - Calls standard setCurrentRecord procedure in data layer.
 - Ensures record is unique
 - Handles commit control
 - Handles data translation issues
 - Processes data translation synchronization.

setLocalRecord

Maintenance Template (edt)

- ▲ genEditCenter
 - Outputs main page content.
 - Customized by developer.
 - Use standard CSS objects for fields.
 - putInputText
 - putCheckBox
 - putTextArea
 - others...

genEditCenter

Maintenance Template

- ▲ Creating a new maintenance program
 - 1. Copy templates
 - 2. Make modifications
 - 3. Setting up CSS
 - a. System Module Maintenance
 - b. Menu Maintenance
 - c. Primary Index Maintenance
 - 4. Testing your code

Create New Maintenance Program

Note Required Javascript libraries in Mnt programs:

standard1.js, standard2.js, maint_nav.js

Required Javascript libraries in Edt programs

standard2.js, maint_edt.js

Case Study – Lab 3

- ▲ Do Lab 3 of Chapter 7 Lab Exercises found in Appendix A

Case Study, Lab 3

Report Template

- ▲ Built from base template
- ▲ Two programs required for a maintenance module (**rpe** and **rpt**)
- ▲ In-line instructions
- ▲ Quick with override points
- ▲ Still developing / enhancing (version 4.1)

Report Template

Overview

Report Template (rpe)

- ▲ Additional features in rpe template
 - genLeftReport
 - genCenterReport
 - genRightReport
 - createInitParameters

Additional Features in rpe Template

Report Template (rpe)

- ▲ genCenterReport
 - Very similar to the genCenterMaintenance
 - putInputText
 - putTextArea
 - putCheckBox

genCenterReport

Report Template (rpe)

- ▲ createInitParameters
 - Populates the form temp-table with initial report parameters.
 - Allows for customization.
 - Code area for custom report restrictions.

createInitParameters

Report Template (rpt)

- ▲ Additional features in rpt template
 - insertReportFooter
 - Replaces standard CSS page footer
 - Better for printing
 - Allows special customization without interfering with standard html page footers.
 - insertReportHeader
 - Replaces standard CSS page header
 - Better for printing
 - Allows special customization without interfering with standard html page headers.

Additional Features in rpt Template

Report Template

- ▲ Creating a new report program
 - 1. Copy templates
 - 2. Make modifications
 - 3. Setting up CSS
 - a. System Module Maintenance
 - b. Menu Maintenance
 - c. Primary Index Maintenance
 - 4. Testing your code

Create New Report Program

Note Required Javascript libraries in Rpe programs:

standard1.js, standard2.js, maint_edt.js

Required Javascript libraries in Rpt programs:

report.js

Case Study – Lab 4

- ▲ Do Lab 4 of Chapter 7 Lab Exercises found in Appendix A

Case Study, Lab 4

Core Objects

- ▲ Common Features
 - Dynamic in Nature
 - Simple Standard Call
 - Encapsulated Functionality
- ▲ Objects Include:
 - Browse
 - Calendar
 - Event
 - Session Management
 - Registry
 - Process Error

Core Objects

Common Features

Core Objects

- ▲ Browse Object
- ▲ Adding a Browse
 - Create browse using Browse Maintenance
 - Module Maintenance – (assign browse.js)
 - Add browse name to putInputText
 - Test Code

Browse Object

Core Objects

- ▲ Calendar Object
- ▲ Adding a Calendar
 - No maintenance needed
 - Module maintenance – calendar.js
 - Add Calendar to putInputText
 - Test Code

Calendar Object

Core Objects

- ▲ Event Object
- ▲ Adding an Event
 - email maintenance
 - Calling processEvent
 - Test Code

Event Object

Allows for different types of events. Currently only supporting email events. You need to set up your email server connectivity in system control before trying to set up an email event.

Core Objects

- ▲ Session Object
- ▲ Utilizing the Session Object
 - wpro_session_value
 - setKeyValue()
 - getKeyValue()
 - Test Code

Session Object

Core Objects

- ▲ Registry Object
- ▲ Utilizing the CSS Registry
 - wpro_registry
 - getValue()
 - setValue()
 - Test Code

Registry Object

Core Objects

- ▲ ProcessError Object
- ▲ Setting Up and Error Message
 - Error Message Maintenance
 - Calling processError
 - Test Code

ProcessError Object

Case Study – Lab 5

- ▲ Do Lab 5 of Chapter 7 Lab Exercises found in Appendix A

Case Study, Lab 5

Move to Supers

- ▲ New Procedures
- ▲ New Functions
- ▲ Add New Layer to the Stack
 - Update Extensions
 - Update Master Include for Extensions
 - Create *super.p's*
 - Update super's stack in Super Layer Maintenance
- ▲ Move Custom Procedures into New Super Layer

Move to Supers

Overview

Creating a New Layer

1. Obtain custom copy of `lib_ext_layerui.i` and `lib_ext_layerex.i`
2. Add reference to `ext_xx_superui.i` and `ext_xx_superex.i` in the master extension layer includes
3. Create a empty .i for the two ext placeholders
4. Create a `lib_xx_superui1.p` and `lib_xx_superex1.p` and include correct super layer reference
5. Go to Super Layer Maintenance and add `lib_xx_superui1.p` and `lib_xx_superex1.p` to the end of the respective stacks
6. Restart the web agents

Creating New Layer

Move Custom Procedures

1. Identify what procedures need to be moved and whether it belongs in the UI or EX layer.
2. Resolve separation of UI / Business Rules / Data
3. Identify shared temp-tables
4. Move one at a time
5. Test each move!

Move Custom Procedures

Case Study – Lab 6

- ▲ Do Lab 6 of Chapter 7 Lab Exercises found in Appendix A

Case Study, Lab 6

Customizing Your Procedures

- ▲ Find the procedure needing to be overridden in the correct ext program.
- ▲ Add custom code before, after, or instead of the RUN SUPER call.
- ▲ Stop and restart web agents.
- ▲ Test Code

Customizing Your Procedures

Case Study – Lab 7

- ▲ Do Lab 7 of Chapter 7 Lab Exercises found in Appendix A

Case Study, Lab 7

Case Study – Lab 8

- ▲ Do Lab 8 of Chapter 7 Lab Exercises found in Appendix A

Modifying Core Product

Case Study, Lab 8

APPENDIX A

Lab Activities

Chapter 3 Labs

CSS Registry Lab

The charts listed on the next page represent tables in your CSS database. Using the information provided in the charts, fill in the correct final value for each case in the following table:

What's the answer?

User Id	Customer #	Key	Answers
Roger	100	CreditLimit	
Roger	200	CreditLimit	
Roger	300	CreditLimit	
Bruce	100	CreditLimit	
Bruce	400	CreditLimit	
Jim	500	CreditLimit	
Bill	300	ShowDate	
Mike	200	ShowDate	
Larry	600	PostOrder	

CSS Registry Lab, cont.

Registry Table

Id	Key	Key Value	Override
	CreditLimit	Yes	Yes
100	CreditLimit	No	Yes
200	CreditLimit	Yes	Yes
500	CreditLimit	No	Yes
Admin	CreditLimit	Yes	No
B2C	CreditLimit	No	Yes
SouthEast	CreditLimit	Yes	Yes
Bruce	CreditLimit	No	Yes
Mike	CreditLimit	No	Yes
	ShowDate	Yes	Yes
Jim	ShowDate	No	No
Manager	ShowDate	Yes	Yes
100	PostOrder	No	Yes

Customer Table

Customer Number	Customer Groups
100	
200	
300	B2C
400	SouthEast
500	B2C,SouthEast

User Table

User Id	Customer Association	Security Group
Roger	100	It
Roger	200	It
Roger	300	It
Bruce	100	Admin
Bruce	400	It
Jim	500	Customer
Bill	300	Customer
Mike	200	Manager
Larry	100	Manager

CSS Registry Lab Answers

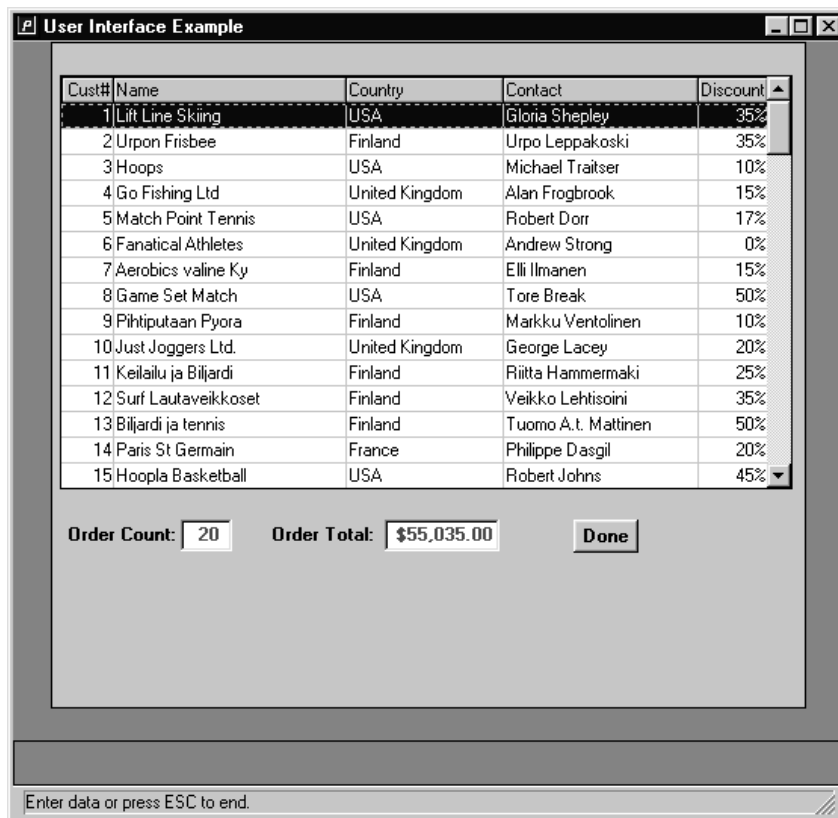
Answers

User Id	Customer #	Key	Path	Answers
Roger	100	CreditLimit	Yes,-,No,-	No
Roger	200	CreditLimit	Yes,-,-,Yes,-	Yes
Roger	300	CreditLimit	Yes,No,-,-,-	No
Bruce	100	CreditLimit	Yes,-,Yes=>	Yes
Bruce	400	CreditLimit	Yes,No,-,-,No	No
Jim	500	CreditLimit	Yes,No,Yes,No	No
Bill	300	ShowDate	Yes,-,-,-,-	Yes
Mike	200	ShowDate	Yes,-,Yes,-,-	Yes
Larry	600	PostOrder	=> None	“”

Chapter 5 Labs

Lab-5a

The following screen was produced from **ui-procx.p**. Open the procedure and save it as **ui-proc.p**.



Cust#	Name	Country	Contact	Discount
1	Lift Line Skiing	USA	Gloria Shepley	35%
2	Urpon Frisbee	Finland	Urpo Leppakoski	35%
3	Hoops	USA	Michael Traitser	10%
4	Go Fishing Ltd	United Kingdom	Alan Frogbrook	15%
5	Match Point Tennis	USA	Robert Dorr	17%
6	Fanatical Athletes	United Kingdom	Andrew Strong	0%
7	Aerobics valine Ky	Finland	Elli Ilmanen	15%
8	Game Set Match	USA	Tore Break	50%
9	Pihtiputaan Pyora	Finland	Markku Ventolinen	10%
10	Just Joggers Ltd.	United Kingdom	George Lacey	20%
11	Keilailu ja Biljardi	Finland	Riitta Hammermaki	25%
12	Surf Lautaveikkokset	Finland	Veikko Lehtisoini	35%
13	Biljardi ja tennis	Finland	Tuomo A.t. Mattinen	50%
14	Paris St Germain	France	Philippe Dasgil	20%
15	Hoopla Basketball	USA	Robert Johns	45%

Order Count: 20 Order Total: \$55,035.00 Done

Enter data or press ESC to end.

- 1 Create a structured procedure called **sessionLib1.p** that contains two internal procedures. One internal procedure should calculate the total number of orders per customer and the other should calculate the total amount of orders per customer.
- 2 Run **sessionLib1.p** persistently from **ui-proc.p** and bind it as a super procedure to the session.
- 3 When the value-changed event occurs in the browser, use the two internal procedures to produce the necessary values for “Order Count” and “Order Total”.

Lab-5b

Notice the different value for the “Order Total” in the screen below.

The screenshot shows a window titled "User Interface Example". Inside, there is a table with the following data:

Cust#	Name	Country	Contact	Discount
1	Lift Line Skiing	USA	Gloria Shepley	35%
2	Urpon Frisbee	Finland	Urpo Leppakoski	35%
3	Hoops	USA	Michael Traitser	10%
4	Go Fishing Ltd	United Kingdom	Alan Frogbrook	15%
5	Match Point Tennis	USA	Robert Dorr	17%
6	Fanatical Athletes	United Kingdom	Andrew Strong	0%
7	Aerobics valine Ky	Finland	Elli IImanen	15%
8	Game Set Match	USA	Tore Break	50%
9	Pihtiutaan Pyora	Finland	Markku Ventolinen	10%
10	Just Joggers Ltd.	United Kingdom	George Lacey	20%
11	Keilailu ja Biljardi	Finland	Riitta Hammermaki	25%
12	Surf Lautaveikkoset	Finland	Veikko Lehtisoini	35%
13	Biljardi ja tennis	Finland	Tuomo A.t. Mattinen	50%
14	Paris St Germain	France	Philippe Dasgil	20%
15	Hoopla Basketball	USA	Robert Johns	45%

Below the table, there are two summary fields:

Order Count: Order Total:

At the bottom of the window, there is a status bar that reads: "Enter data or press ESC to end."

- 1 Create a new procedure called **extLib1.p** that will contain an internal procedure with the same name as the one in **sessionLib1.p** that calculates the total amount of a customer's orders.
- 2 Run **extLib1.p** persistently but bind it as a super procedure to **ui-proc.p**.
- 3 Without changing the logic that exists in the session super procedures, cause the customer's discount rate to be applied to the "Order Total" value.

Lab-5c

Create a local copy of the sports or sports2000 database supplied with PROGRESS. Create a WebSpeed broker so that it's agents connect at startup to your database.



The screenshot shows the 'Dynamic Query Lab - Microsoft Internet Explorer' window with the query results displayed as a table. The address bar displays 'http://localhost/scripts/cgiip.exe/WService=wsbroker1/dynqry1.r'. The table has the following columns: CustNum, Name, City, State, PostalCode, and Phone. The status bar at the bottom shows 'Done' and 'Local intranet'.

CustNum	Name	City	State	PostalCode	Phone
1	Lift Line Skiing	Boston	MA	01730	(617) 450-0086
2	Urpon Frisbee	Oslo	Uusima	45321	(603) 532 5471
3	Hoops	Atlanta	GA	02445	(617) 355-1557
4	Go Fishing Ltd	Harrow	Middlesex	HA8 bbb	081 883 6827
5	Match Point Tennis	Boston	MA	02134	(817) 498-2801
6	Fanatical Athletes	Montgomery	AL	48855	0224 692 903
7	Aerobics valine Ky	Tikkurila	Uusimaa	4567	(90) 054 6399
8	Game Set Match	Deatsville	AL	851 14	060-60 62 61
9	Pihtiutaan Pyora	Pihtiupudas	Etela-Savo	44800	(60) 162 5007
10	Just Joggers Ltd.	Ramsbottom	Sussex	BL0 9ND	070 682 2887
11	Keilailu ja Biljardi	Helsinki	Uusimaa	210	(91) 972 3478

Lab-5d

Copy the **dynqry.html** file from the Labs subdirectory into your working directory. Modify it so when the “Run Query” button is pressed a call is made to a second web object that will take the table and field list values and create an HTML table displaying the first 20 records.

Since any table and field combinations can be selected, the secondary web object must utilize dynamic objects. Save your objects as **dynqry1.html** and **dynqry2.html**.

Lab-5e

Modify **dynqry2.html** so that it uses a dynamic temp-table. Create a new Progress procedure where the creation of the dynamic temp-table will be handled. After all the desired records are read into the dynamic temp-table, pass the handle of the dynamic temp-table back to the secondary web object **dynqry2.html**. Use the dynamic temp-table as the basis for the dynamic query in **dynqry2.html** and produce the HTML table as before.

Chapter 6 Labs

Naming Conventions

Follow the steps below to create a progress procedure and its various components. There is no need to create any programming logic or special functionality since this is simply an exercise designed to demonstrate the CSS naming conventions.

- 1** Create the shell of a new Progress procedure. Treat it as if it were a new order processing module (op).
- 2** Save your new procedure in the proper directory structure.
- 3** Modify the new procedure so that it contains two internal procedures.
 - a** Set up the first procedure as if its purpose was to output some html text.
 - b** Set up the second procedure as if its purpose was to return some data.
- 4** Place a standard CSS temp-table definition in your new procedure. Add the following fields:
 - a** A character string field.
 - b** An integer field.
 - c** A handle to some object.
 - d** A field used to store a rowid.
- 5** Modify the first internal procedure so that it calls the second procedure that returns data. Have the two procedures pass the temp-table created in step 4 between them.

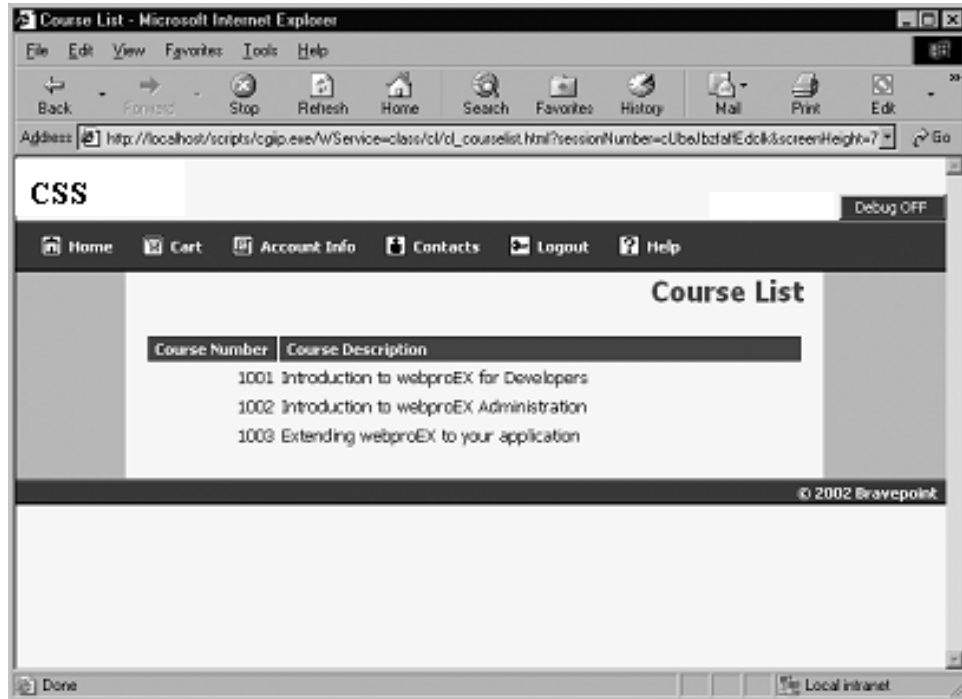
Chapter 7 Labs

Set-up Instructions

- 1 Create a modifications directory called “class” (*drive_letter:/class*).
- 2 Copy the templates directory from the courseware CD to the modifications directory.
- 3 Modify the CSS.ini file:
 - a Add the “class” directory to the PROPATH settings.
 - b Add the database connect statement for the class database.
 - c Save changes to the CSS.ini file.
- 4 Stop and restart the class WebSpeed broker.

Case Study - Lab 1

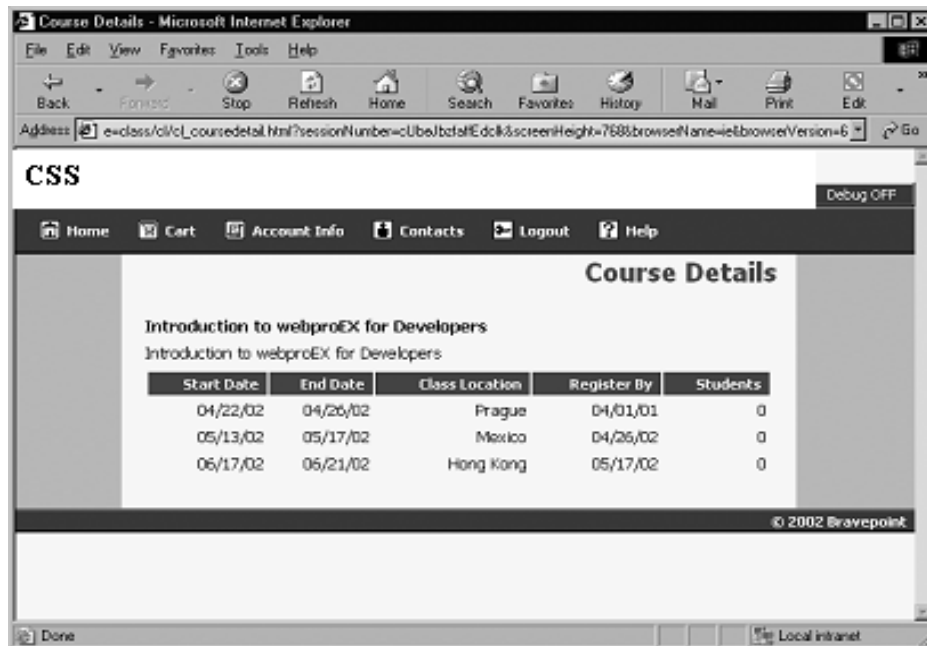
Create a new procedure from the basic template. It should display a list of courses from the wpro_course database table. Save the procedure as **cl_courselist.html**.



Case Study - Lab 1 (continued)

Create a second procedure from the basic template. It should display a list of classes for a particular course. It will access data from both the wpro_course table and the wpro_class table. Save the procedure as **cl_coursedetail.html**.

In the **cl_coursedetail.html** procedure, create an internal procedure that will calculate the number of students in a particular class. The logic should exclude students if their title begins with "CSS". Set up the internal procedure so it is passed a class number as an input parameter and returns the number of students as an output parameter.



Note Instructions for completing this lab are found on the next page.

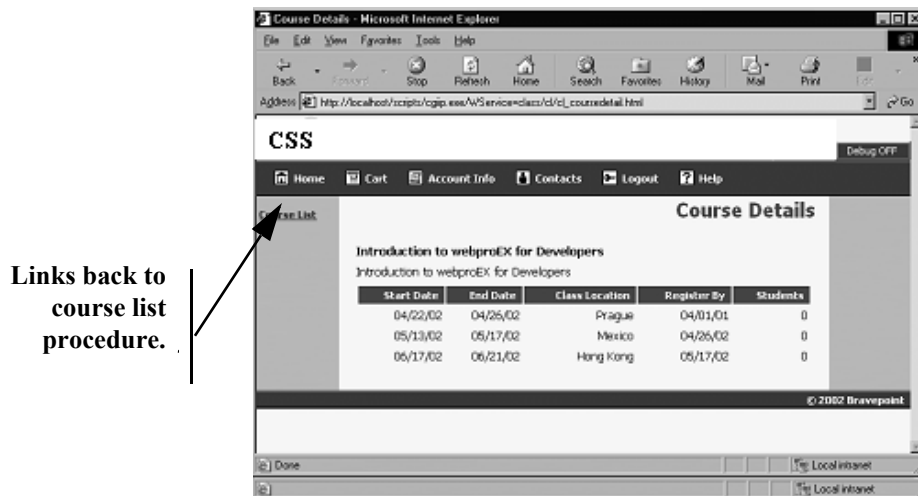
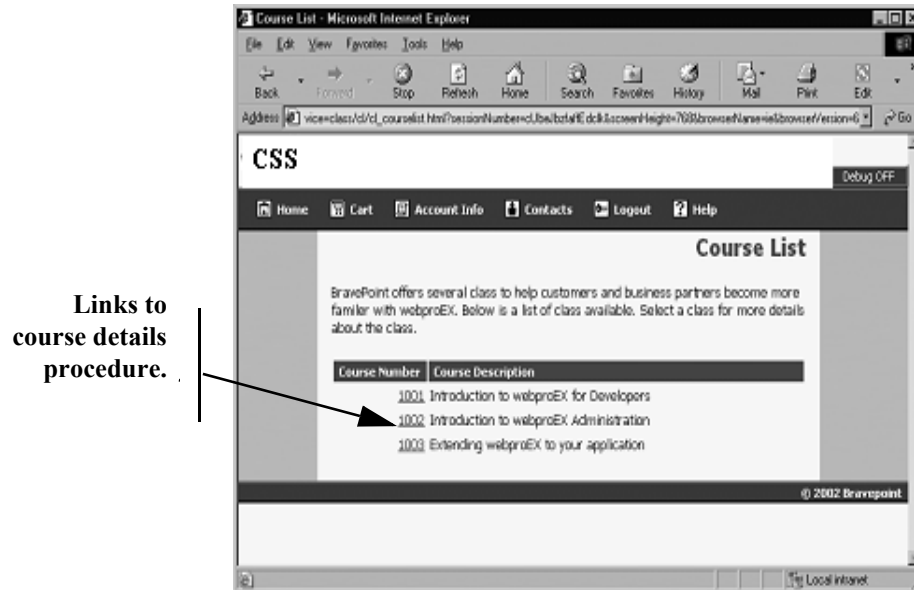
Case Study - Lab 1 (continued)

Use the following instructions to help create the basic template procedures:

- 1** Copy the base template to the custom module directory and rename it.
- 2** Create an include file that contains a temp-table definition for the database table. Be sure to use proper naming conventions.
- 3** Edit the new template / procedure:
 - a** Change program name parameter in lib/lib_sys_std.i include
 - b** Insert include file with temp-table definition for wpro_course
 - c** Rename internal procedure “**genCenterXXX**” to a meaningful name (i.e., “**genCourseCenter**”). Update the internal procedure call to reflect the new name.
 - d** Add code in new procedure to create screen output.
 - e** Compile code and check for errors.
- 4** Log into CSS to set up new module.
- 5** Go to the menu selection:
Administration – Control Menu – System Module Maintenance.
- 6** Press “Add” and fill in the form fields to add new module (see Admin Guide for additional information)
- 7** After new module has been added, find newly added module and click on “view” link in Javascript column.
- 8** Add the standard1.js file reference.
- 9** Go back to the home screen and go to menu selection:
Administration – Control Menu – Select Security Group.
- 10** Press the “Add New Header” button.
- 11** Rename the new header entry to “Class” (This only needs to be done once.)
- 12** Press the “Save” button.
- 13** Click on the newly added module in the module list.
- 14** Move the module into the “Class” header.
- 15** Go to Home.
- 16** Find menu selection and run new program.

Case Study - Lab 2

Connect the two programs created in case study Lab 1 (cl_courselist.html and cl_coursedetail.html) so you can navigate and post between them.



Note Instructions for completing this lab are found on the next page.

Case Study - Lab 2 (continued)

Use the following instructions to help link `cl_courselist.html` and `cl_coursedetail.html`:

Edit `cl_courselist.html` as follows:

- 1 Create a Javascript function called "preSubmit()" in `cl_courselist.html`.
- 2 Create a hidden form field called `kCourseNumber` in the center form.
- 3 Create an html anchor tag around the course numbers using the following code:
{&OUT} "".

Edit `cl_coursedetail.html` as follows:

- 1 Create a Javascript function called "preSubmit()" in `cl_coursedetail.html`.
- 2 Use the GET-VALUE webspeed API to obtain the passed course number `kCourseNumber`.
- 3 Create an internal procedure called `genCourseDetailLeft`.
- 4 Place an anchor tag inside `genCourseDetailLeft` to point back to `cl_courselist.html`. Use the following code:

**{&OUT} "
" getXlate("Course List") "" skip.**

preSubmit() in cl_courselist.html:

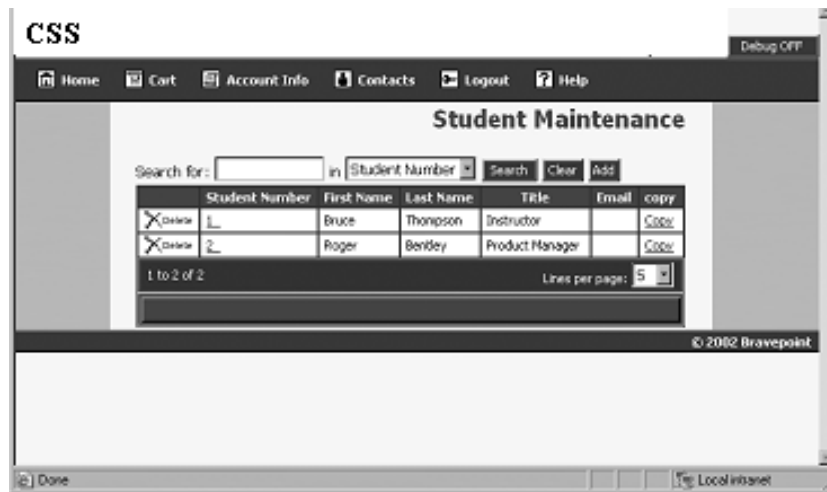
```
function preSubmit(objValue){
    document.centerForm.kCourseNumber.value = objValue;
    document.centerForm.action = "../cl/cl_coursedetail.html";
    document.centerForm.submit();
}
```

preSubmit() in cl_coursedetail.html:

```
function preSubmit(){
    document.centerForm.action = "../cl/cl_courselist.html";
    document.centerForm.submit();
    return;
}
```

Case Study - Lab 3

Create maintenance programs for the following four tables: course (wpro_course), class (wpro_class), student (wpro_student), classRegistration (wpro_class_registration). Be sure to include the primary index and other index fields in the navigation. The edit programs (edt) should allow for the editing of all fields.



The screenshot shows the 'Student Edit - Microsoft Internet Explorer' window with a 'Detail Student Edit' form. The form contains the following fields:

- Student Number: 1
- First Name: Bruce
- Last Name: Thompson
- Title: Instructor
- Address: 5875 Peachtree Ind Blvd
- Suite: Suite 200
- City: Norcross
- State: GA
- Postal Code: 30092
- Home Phone: 770 111 2222
- Work Phone: 770 449 9696
- Email: (empty field)

At the bottom of the form are 'Save' and 'Cancel' buttons.

Note Instructions for completing this lab are found on the next two pages.

Case Study - Lab 3 (continued)

Use the following instructions to help create the student maintenance programs.

- 1** Copy the two template programs (mnt and edt) into your custom modules directory.
- 2** Rename the copies. (i.e., ad_mnt_template.html -> cl_mnt_student.html)
- 3** Create the temp-table definition for the database table and save it in a properly named include file.

Maintenance Procedure (mnt)

- 1** Edit the mnt file following the in-line steps (cl_mnt_student.html):
 - a** Step 1: change the program name.
 - b** Step 2: change the temp-table include reference.
 - c** Step 3: change the temp-table preprocessor directive.
 - d** Step 4: set list of fields to be used in search and sort.
 - e** Step 5: set search values list.
 - f** Step 6: set search field labels
 - g** Step 7: set list of fields for display and order.
 - h** Step 8: set list of displayed field labels.
 - i** Step 9: supply the physical database table name.
 - j** Step 10: supply the edt program name.
 - k** Step 11: supply the field used for the anchor tag.
 - l** Step 12: supply the name of any child tables involved in the maintenance.
 - m** Step 13: supply the label to the child table link.
 - n** Step 14: supply the name of the program to return to if child table maintenance is used.
 - o** Step 15: set the values for Javascript pop-up window.
 - p** Step 16: edit any custom logic needed for validating or cleaning up before or after a delete operation.
 - q** Step 17: adjust code for parent / child join. Comment out if not used.
- 2** Save and compile code.
- 3** Follow standard steps for adding modules to CSS menus.
- 4** Perform Primary Index Maintenance.

Case Study - Lab 3 (continued)

Edit Procedure (edt)

- 1** Edit the edt file and follow the in-line steps (cl_edt_student.html):
 - a** Step 1: change the program name.
 - b** Step 2: change the temp-table include reference.
 - c** Step 3: change the temp-table preprocessor directive.
 - d** Step 4: set the preprocessor to the name of the temp-table.
 - e** Step 5: set the name of the physical database table.
 - f** Step 6: set the variable to the name of the temp-table.
 - g** Step 7: enter required fields in comma delimited list.
 - h** Step 8: build the html code for data output using appropriate api's.
- 2** Save and compile code.
- 3** Follow standard steps for adding modules to CSS menus.

Case Study - Lab 4

Create four reports using the report templates. Create one set of procedures (rpe and rpt) for each of the four database tables.

Class Code	Course Code	Start Date	End date	Location	Close Out Date
001	1001	04/22/02	04/26/02	Prague	04/01/01
002	1001	05/13/02	05/17/02	Mexico	04/26/02
003	1001	06/17/02	06/21/02	Hong Kong	05/17/02

Note Instructions for completing this lab are found on the next page.

Case Study - Lab 4 (continued)

Use the following instructions to help create the report programs.

- 1** Copy the two template programs (rpe and rpt) into your custom modules directory.
- 2** Rename the copies. (i.e., ad_rpe_template.html -> cl_rpe_class.html)

Report Entry Procedure (rpe)

- 3** Edit the rpe file following the in-line steps (cl_rpe_class.html):
 - a** Step 1: Change the program name.
 - b** Step 2: Set the temp-table name. Be sure to use proper naming convention for custom tables. The temp-table definition must include all the parameters being passed to the report body (rpt).
 - c** Step 3: Set preprocessor directive to temp-table name.
 - d** Step 4: Set the variable to the report body program name (rpt).
 - e** Step 5: Set the list of report fields and their order to pass to the report body.
 - f** Step 6: Set variable to delimited list of required fields.
 - g** Step 7: Build body with genCenterReport api.

Report Body Procedure (rpt)

- 4** Edit the rpt file following the in-line steps (cl_rpt_class.html):
 - a** Step 1: Change the program name.
 - b** Step 2: Include the temp-tables that are needed.
 - c** Step 3: Define any variables needed.
 - d** Step 4: Create temp-table record for passed report parameters.
 - e** Step 5: Build query string using parameters in temp-table and call dataEngine.
- 5** Save and compile code.
- 6** Follow standard steps for adding modules to CSS menus.

Case Study - Lab 5

Browse Objects

Create four browse objects using the Browse Maintenance form. Create a “From” and “To” for both the course and class fields. Use the following information to fill out the form.

After you have created all four browse objects, go through the Module Maintenance menu function and add the browse.js Javascript file to **cl/cl_rpe_class.html**.

Edit **cl/cl_rpe_class.html** and add the browse object names using **putInputText** to the appropriate fields.

Detail Edit Browse Master us

wproVersion: ex 4.0

Browse Module: cl/cl_rpe_class.html

Browse Name: ClassFrom

Browse UserID:

Browse Title: Class List

Table Name: wpro_class

Query string for table:

Key search field for table: *,,,,

Field type of Key: Char

Table sort expression:

List of labels: Class Code,Course Code,Start Date,Location,Close Out

List of fields: wpro_class.wpro_class_code,wpro_class.course_code,wpro

Browse return list: centerForm.hf_rpp_class_from

Additional fields program:

Override program name:

Save Cancel

List of Fields:

- wpro_class.wpro_class_code,
- wpro_class.course_code,
- wpro_class.class_start_date,
- wpro_class.class_location,
- wpro_class.class_closeout_date

Case Study - Lab 5 (continued)

Edit Browse Master us - Microsoft Internet Explorer

Detail Edit Browse Master us

wproVersion: ex 4.0

Browse Module: cl/cl_rpe_class.html

Browse Name: ClassTo

Browse UserID:

Browse Title: Class List

Table Name: wpro_class

Query string for table:

Key search field for table: ,* ,,,,

Field type of Key: Char

Table sort expression:

List of labels: Class Code,Course Code,Start Date,Location,Close Out

List of fields: wpro_class.wpro_class_code,wpro_class.course_code,wpro_

Browse return list: ,centerForm.hf_rpp_class_to

Additional fields program:

Override program name:

List of Fields:

- wpro_class.wpro_class_code,
- wpro_class.course_code,
- wpro_class.class_start_date,
- wpro_class.class_location,
- wpro_class.class_closeout_date

Case Study - Lab 5 (continued)

Edit Browse Master us - Microsoft Internet Explorer

Detail Edit Browse Master us

wproVersion:

Browse Module:

Browse Name:

Browse UserID:

Browse Title: 

Table Name:

Query string for table:

Key search field for table:

Field type of Key:

Table sort expression:

List of labels: 

List of fields:

Browse return list:

Additional fields program:

Override program name:

List of Fields:

- wpro_course.course_code,
- wpro_course.course_name,
- wpro_course.course_days,
- wpro_course.course_price,
- wpro_course.course_short_desc

Case Study - Lab 5 (continued)

Edit Browse Master us - Microsoft Internet Explorer

Detail Edit Browse Master us

wproVersion: ex 4.0

Browse Module: cl/cl_rpe_class.html

Browse Name: CourseTo

Browse UserID:

Browse Title: Course List

Table Name: wpro_course

Query string for table:

Key search field for table: ,*,,,,

Field type of Key: Char

Table sort expression:

List of labels: Course Code,Course Name,Length,Price,Description

List of fields: wpro_course.course_code,wpro_course.course_name,wpro_

Browse return list: centerForm.hf_rpp_course_to

Additional fields program:

Override program name:

Save Cancel

List of Fields:

- wpro_course.course_code,
- wpro_course.course_name,
- wpro_course.course_days,
- wpro_course.course_price,
- wpro_course.course_short_desc

Case Study - Lab 5 (continued)**Calendar Objects**

Add calendar objects to the Class Report Entry screen (cl/cl_rpe_class.html). Position one calendar object next to each input date field in the form.

- 1 Go to Module Maintenance and add the calendar.js Javascript file to cl/cl_rpe_class.html.
- 2 Edit cl_rpe_class.html and add the calendar option to putInputText.

The finished product should look similar to the following:

The screenshot shows a web browser window titled "Class Report - Microsoft Internet Explorer". The address bar displays a local URL. The page layout includes a top navigation bar with links for Home, Cart, Account Info, Contacts, Logout, and Help. Below this is a "Class Report" section containing three rows of input fields. The first row is labeled "Class From:" and "To:", the second "Course From:" and "To:", and the third "Class Start Date From:" and "To:". Each "To:" field has a small calendar icon next to it. A "Submit" button is positioned below the input fields. The footer of the page indicates "© 2002 Bravepoint".

Case Study - Lab 5 (continued)

Event Object

Set up an email event using **Email Maintenance**. Set the event to send an email when the class report has finished. Modify the Class Report code to process the email.

- 1** Go to **Admin – Message Maintenance – Email Master Maintenance** and set up the email event.
- 2** Edit **cl_rpt_class.html** and insert the processEvent API call.

Session Object

Utilizing Session Management, show the user special instructions on running the class report from the report entry screen. Keep showing the instructions until the user has submitted the report. Add setValue() and getKeyValue() to their proper location in the code.

Registry Object

Using a registry flag, control whether or not the class report shows the end date of a class or the number of days the class lasts.

- 1** Go to **Registry Maintenance**. Set up a new key.
- 2** Set the value of the new key.
- 3** Edit your code to access the new Registry Key using the getValue() API.

Process Error Object

When the user submits a class report and the parameters result in no found records, utilize the setProcessError() API to write a message to the webspeed server log.

Case Study - Lab 6

- 1 Create a new layer and move your custom EX and UI elements into the correct location.
- 2 Follow the steps previously demonstrated in the lecture to accomplish this.
- 3 Use the “cl” directory for the module name.

Case Study – Lab 7

- 1 Using the ext, override the procedure that calculates the number of students attending each class.
- 2 The original procedure excluded any student from the count if their title began “CSS”. Override this logic so that all students are counted.

Case Study – Lab 8

- 1 Override the display of the CSS version number on the main screen with a message welcoming the user by name.
- 2 Extend the standard CSS footer to include a company address below the current footer bar.
- 3 Use a registry flag to store the data for the address.

Next, you will be introduced to the CSS layers and how they interact with each other. We will explain CSS’ data architecture as well as the CSS superstructures or super-procedure libraries.

APPENDIX B

Additional CSS APIs

dataEngine

Purpose: Collects data dynamically and returns the required subset to the UI element in the form of a populated temp-table.

API: RUN dataEngine (INPUT *table name*,
 INPUT *queryWhere*,
 INPUT *cRowid*,
 INPUT *cAction*,
 INPUT *cMode*,
 INPUT *iSetCnt*,
 INPUT *cSession*,
 INPUT *cFunct*,
 INPUT *rtnTotal*,
 INPUT *cJump*,
 INPUT *outTable*,
 OUTPUT *cLabel*,
 OUTPUT *cJump*,
 OUTPUT *iCount*,
 OUTPUT *kErrorMessage*,
 OUTPUT *cTotalRecords*,
 OUTPUT *hdio*).

Return Value: None

Parameters:

Input:	<i>cTableName</i>	AS CHARACTER – actual db table name
	<i>queryWhere</i>	AS CHARACTER – query phrase or expression
	<i>cRowid</i>	AS CHARACTER – rowid of record if desired
	<i>cAction</i>	AS CHARACTER – query action, usually “first”
	<i>cMode</i>	AS CHARACTER – record lock mode
	<i>iSetCnt</i>	AS INTEGER – maximum number of records in set
	<i>cSession</i>	AS CHARACTER – user Session Number
	<i>cFunct</i>	AS CHARACTER – current page (i.e., <i>kCurPage</i>)
	<i>rtnTotal</i>	AS CHARACTER – total number of records
	<i>cJump</i>	AS CHARACTER – record number in set to jump to
	<i>outTable</i>	AS CHARACTER – actual db table name
Output:	<i>cLabel</i>	AS CHARACTER – obsolete but required by API
	<i>iCount</i>	AS INTEGER – obsolete but required by API
	<i>kErrorMessage</i>	AS CHARACTER – obsolete but required by API
	<i>cTotalRecords</i>	AS INTEGER – obsolete but required by API
	<i>hdio</i>	AS TABLE-HANDLE – handle to temp-table

genNavigation

Purpose: Element that controls the number of records to be displayed in a browse. This works in conjunction with the data engine.

API: RUN genNavigation (INPUT *iJumpAmount*,
INPUT *iRecordsInSet*,
INPUT *kTotalRecords*,
INPUT *isLargeTableBrowse*,
INPUT *kcolspan*)

Return Value: None

Parameters:

Input: <i>iJumpAmount</i>	AS INTEGER – jump amount
<i>iRecordsInSet</i>	AS INTEGER – number of records in set
<i>kTotalRecords</i>	AS CHARACTER – total number of records
<i>isLargeTableBrowse</i>	AS LOGICAL – large table browse?
<i>kcolspan</i>	AS INTEGER – column span value

getDatalate

Purpose: To get the translated data value of CSS database fields.

API: GetDatalate(INPUT *kTableName*,
INPUT *kFieldName*,
INPUT *kLang*,
INPUT *kFieldValue*,
INPUT *kPUK*)

Return Value: Translated string

Parameters:

Input: <i>kTableName</i>	AS CHARACTER – name of table to be translated
<i>kFieldName</i>	AS CHARACTER – name of field in table to be translated
<i>klang</i>	AS CHARACTER – Language of return string
<i>kFieldValue</i>	AS CHARACTER – The current value of the field to be translated.
<i>kPUK</i>	AS CHARACTER – The concatenated value of the fields which comprise the primary unique index of the record requiring translation. (PUK)

getKeyValue()

Purpose: Returns the required session key value stored by setKeyValue.

API: `getKeyValue(INPUT kSessionNum, INPUT kKey)`

Return Value: Character

Parameters:

Input: kSessionNum AS CHARACTER – user’s session number
kKey AS CHARACTER – key value desired

getPhraselate()

Purpose: To translate specified string into other language and replace run time values in the string.

API: getPhraselate(INPUT *kSessionNumber*,
 INPUT *kUserLang*,
 INPUT *kString*,
 INPUT *ParamVars*,
 INPUT *ParamValues*)

Return Value: Character

Parameters:

Input:	kSessionNumber	AS CHARACTER– The current session identifier of the user
	kUserLang	AS CHARACTER– The current language key of the user
	kString	AS CHARACTER– The text string to be translated
	kParamVars	AS CHARACTER– The replacement symbols to be replaced by the kParamValues in the final translated text.
	kParamValues	AS CHARACTER - The values of the replacement symbols to be replaced in the final translated text

getPUK

Purpose: To get the PUK values from the database for given table/field. Used by getDateLate().

API: `getPUK( INPUT table name, INPUT RowID)`

Return Value: Value of the PUK

Parameters:

```
Input:  kTableName AS CHARACTER
        kROWID      AS ROWID
```

getValue() (UI layer)

Purpose: Gets the value of a user input field from the CSS registry.
API: getValue(INPUT *cInput*)
Return Value: Character
Parameters:
Input: *cInput* AS CHARACTER – input field name

getValue (EX layer)

Purpose: Gets the value of a registry key from the CSS registry.
API: RUN getValue(INPUT *c1*, INPUT *c2*, OUTPUT *c3*)
Return Value: None
Parameters:
Input: *c1* AS CHARACTER – registry key name
c2 AS CHARACTER – users session number
Output: *c3* AS CHARACTER – registry key value

getXlate()

Purpose: To get the translated value of CSS strings.
API: getXlate(INPUT *kString*)
Return Value: Character
Parameters:
Input: *kString* AS CHARACTER - *string to be translated*

processEvent

Purpose: This procedure is the main processor to send mail. Along with sending the mail it writes to an email log so that in the future an email can be re-sent.
API: RUN processEvent(INPUT *kSessionNumber*,
INPUT *kEventID*,
INPUT *kemevChar1*,
INPUT *kemevChar2*)
Return Value: None
Parameters:
Input: *kSessionNumber* AS CHARACTER - user session number
kEventID AS CHARACTER - event id
kemevChar1 AS CHARACTER - *replacement string*
kemevChar2 AS CHARACTER - *replacement string values*

putBrowse()

Purpose: Standard way to insert a browse to any html page.

API: putBrowse(INPUT *kSessionNumber*,
 INPUT *kImagePath*,
 INPUT *kBrowseName*,
 INPUT *kBrowseModule*,
 INPUT *kOtherJavascript*)

Return Value: Character

Parameters:

Input: kSessionNumber AS CHARACTER – user's session number
 kImagePath AS CHARACTER – path to icon image
 kBrowseName AS CHARACTER – defined browse name
 kBrowseModule AS CHARACTER – defined browse module/ function
 kOtherJavascript AS CHARACTER – additional javascript

putCalendar()

Purpose: Standard way to insert a Calendar pop-up on your HTML page.

API: putCalendar(INPUT *kSessionNumber*,
 INPUT *kImagePath*,
 INPUT *kReturnLocation*,
 INPUT *kAltLabel*)

Return Value: Character

Parameters:

Input: kSessionNumber AS CHARACTER - user's session number
 kImagePath AS CHARACTER – path for the icon image
 kReturnLocation AS CHARACTER – return field (i.e., formName.fieldName)
 kAltLabel AS CHARACTER – alternate field label

putCheckBox

Purpose: A standard function to put a checkbox into the standard maintenance edit template (or any other html page).

API: RUN putCheckBox (INPUT *kCurPage*,
INPUT *kLabel*,
INPUT *iColSpan*,
INPUT *kName*,
INPUT *kValue*)

Return Value: Logical

Parameters:

Input: kCurPage AS CHARACTER – curPage variable stored current page
kLabel AS CHARACTER – Label for check box
iColSpan AS INTEGER – number of table columns to span
kName AS CHARACTER -
kValue AS LOGICAL

putInputText

Purpose: Standard way to insert a text field on your HTML Popup page (or any other html page).

```
RUN putInputText (INPUT kCurPage,
                  INPUT kLabel,
                  INPUT iSize,
                  INPUT kName,
                  INPUT kValue,
                  INPUT kType,
                  INPUT kBrowseCalendar,
                  INPUT kDataLate)
```

API:

Return Value: Logical

Parameters:

Input: <i>kCurPage</i>	AS CHARACTER – The name of the current web page ; the name of the CSS module.
<i>klabel</i>	AS CHARACTER – The label to display for this field.
<i>iSize</i>	AS INTEGER – The size of the fill-in box
<i>kName</i>	AS CHARACTER – The name of this html form field.
<i>kValue</i>	AS CHARACTER – The initial value of this fill-in field.
<i>kType</i>	AS CHARACTER – The data type for this field, i.e., char, date.
<i>kBrowseCalendar</i>	AS CHARACTER – The name of the browse widget to attach to this field.
<i>kDataLate</i>	AS CHARACTER – The PUK value of the field when backed by a database field.

putTextArea

Purpose: Standard way to insert a textarea field on your HTML Popup page (or any other html page).

RUN putTextArea (INPUT *kCurPage*,

INPUT *kLabel*,

INPUT *iSize*,

API: INPUT *kName*,

INPUT *kValue*,

INPUT *kType*)

Return Value: Logical

Parameters:

Input: *kCurPage* AS CHARACTER – The name of the current web page ; the name of the CSS module.

klabel AS CHARACTER– The label to display for this field.

iSize AS INTEGER– The size of the text area.

kName AS CHARACTER– The name of this html form field.

kValue AS CHARACTER – The initial value of this fill-in field.

kType AS CHARACTER– The data type for this field, i.e. char, date, etc.

setKeyValue()

Purpose: Writes the key value into the session variable table. Use the getKeyValue() function to retrieve.

API: setKeyValue(INPUT *kSessionNumber*, INPUT *kKey*, INPUT *kKeyValue*)

Return Value: Logical

Parameters:

Input: *kSessionNum* AS CHARACTER – user's session number

kKey AS CHARACTER – name of key

kKeyValue AS CHARACTER - value to store in key

setProcessError()

Purpose: This function processes error message requests within CSS.

setProcessError(INPUT *kSessionNumber*,

INPUT *kErrorNumber*,

INPUT *kAction*,

API: INPUT *kOverride*

INPUT *kRepString*,

INPUT *kRepValue*)

Return Value: The translated string of the error message.

Parameters:

Input: *kSessionNumber* AS CHARACTER – user's session number
kErrorNumber AS CHARACTER – error number value
kAction AS CHARACTER – Action to take: "html", "email", or "log"
kOverride AS CHARACTER – module name
kRepString AS CHARACTER – symbol replacement string
kRepValue AS CHARACTER – symbol replacement value

Examples:

```
RUN deleteRecord (INPUT sessionNumber,INPUT kMaintTable,INPUT cRowid, OUTPUT
cTotal, OUTPUT kErrorMessage)
```

```
RUN getParentJoin ( INPUT sessionNumber, INPUT curPage + "_TASK_ParentJoin", OUTPUT
KTempJoinPhrase).
```

```
putInputText(INPUT kcurPage, INPUT getXlate("Course Code"),INPUT 8,INPUT 3,INPUT
"hf_course_code",INPUT {%TEMP-TABLE-NAME}.course_code, INPUT "CHAR",INPUT
"",INPUT "{&EDIT-TABLE-NAME}^" + STRING({&TEMP-TABLE-NAME}.tbROWID)).
```

```
putTextArea(INPUT kCurpage, INPUT "40",INPUT "5",INPUT
"hf_course_short_desc",{&TEMP-TABLE-NAME}.course_short_desc, INPUT string(kTable +
"^" + string({&TEMP-TABLE-NAME}.tbsrtOrder))).
```

```
kErrorMessage = setProcessError(sessionNumber,"CL000001","", "", "[@passQuery]",passQuery).
```

```
RUN ProcessEvent (INPUT sessionNumber,
```

```
    INPUT "cl000001",
```

```
    INPUT "[@passQuery]",
```

```
    INPUT passQuery).
```

```
setKeyValue(INPUT sessionNumber, INPUT curPage + "RunOnce", INPUT "Yes").
```


APPENDIX C

Internationalization

Purpose

This project allows CSS to support and serve more than one language concurrently. All label strings, dates, and numbers displayed in html pages are translated accordingly. Translation strings are stored in the database as a translation dictionary. Several functions are used to retrieve data. A customer can purchase CSS International version with a default language set up. Every language can be purchased/loaded separately. The language code is set as soon as the user logs in. The user's language preference is stored in the user's profile. Character set, date format, numeric format are set up according to the language code.

Three topics are covered in this documentation:

Translation

All strings shown in CSS come from different sources. The translation is done in 6 tasks: html page header setup/labels translation, Javascript messages pre-load translation, Progress messages translation, Database fields translation, Images and Multi Currency.

Database Tables

Several database tables require set up for internationalization:

wpro_currency	wpro_data_trans
wpro_error_action	wpro_error_msg
wpro_idx	wpro_string_trans
wpro_lang_control	wpro_carrier_href
wpro_cust_item	wpro_faq
wpro_faqCategory	wpro_info_window_detail
wpro_item_types	wpro_user_master

Images

Translation

Html Page - Header set up

Different languages use different character set in html pages. A META tag is included in every html page to alert the browser to use the specified character set. Since numeric and date fields use different formats depending on the language they are set for the session as soon as the user logs in.

Character Set

Function `putCharSet()` is called to insert META tag of character set in every html page. The data is stored in `wpro_lang_control.character_set`.

Number format

There are two data elements to set: thousands separator and decimal point. The data is stored in `wpro_lang_control.number_format`, first character is the thousands-separator, second character is the fractional-separator. The number format is set in the function `checkSession()`.

`SESSION:SET-NUMERIC-FORMAT (thousands-separator, fractional-separator)`

Note The values of the thousands-separator can NOT be represented by any of the following characters or symbols: B C D R Z z 0 1 2 3 4 5 6 7 8 9 + - < > () * ?, any multi-byte character. The values of the fractional-separator can NOT be represented by any of the following characters or symbols: B C D R Z z 0 1 2 3 4 5 6 7 8 9 + - < > () * ?, the space character, any multi-byte character.

Date format

The date display format, the data is stored in `wpro_lang_control.date_format`.

`SESSION:DATE-FORMAT`

Note Valid values are “mdy”, “dmy”.

Html Page - Labels Translation

The labels in CSS can be static strings, data in database or `wpro_registry` override labels. We'll only look at static strings in this section. The function `getXlate()` is used for the string translation. The static strings with embedded parameter(s) use the function `getPhraselate()`.

Javascript Messages Translation

Javascript messages are stored in the `err_msg.js` file. The files reside in the sub directory of the scripts directory. The name of the sub directory is language code followed by “_msg” (e.g. `scripts/eng_msg`). There is one `err_msg.js` file for every language supported in the system. It is loaded into the html pages by the function `InsertNewJavaScript()`. Every Javascript message is assigned a code. The code is prefixed with “e_” followed by the sequence number, followed by the language type. The sequence numbers are non-leading zero numbers starting with 1. Types can be “msg” for messages, “calm” for calendar month, etc. (e.g. “e_100msg”, “e_11calm”).

Implementation

All Javascript messages are stored in the `wpro_error_msg` table, with the `wpro_type` field set to “js”. A job is run to create `err_msg.js` files (see [Installation Procedure for Business Partner – load.p](#) in section “Create JS files”).

Progress Message Translation

All CSS Progress related error messages are handled in the function `setProcessError()`. Every error message is assigned an error code. The code uses the CSS naming standard: a two-character module code followed by a six-character sequence number with leading zeros.

There are three types of actions for the error:

- html - return the message to be displayed in the html page
- log - output the message to the log file
- email - email the message to a specified email address

Implementation

The error messages are stored in the `wpro_error_msg` table with the `wpro_type` field set to “pro”. The error actions are stored in the `wpro_error_action` table. The `error_action_code` field contains an error code of “html”, “log” or “email”. The `error_action` field stores a logical value of “true” or “false” to indicate an active or inactive error action.

Database Translation

Database tables are evaluated for data language translation. Some tables require the addition of a new language code field as part of the index. When data fields require translation, the function `getDatalate()` is used. The `wpro_registry` table will be handled separately from the other tables. When data from the registry requires translation the function `getDataValue()` is used.

Add Language Code to Index

For database tables with too many fields to translate or data should be separated by language code. Some of these tables are:

<code>wpro_carrier_href</code>	<code>wpro_faqCategory</code>
<code>wpro_cust_item</code>	<code>wpro_info_window_detail</code>
<code>wpro_faq</code>	<code>wpro_item_types</code>

Data Translation

For data stored in database require translation. Use the `getDatalate()` function for the following fields:

Table Name	Table Fields Requiring Translation
<code>wpro_browse_mstr</code>	<code>table_field_labels</code> , <code>browse_title</code>
<code>wpro_customer_care</code>	<code>contact_title</code>
<code>wpro_emev_master</code>	<code>subject</code> , <code>body</code>
<code>wpro_icons</code>	<code>alt_text</code> , <code>image_alt_text</code>
<code>wpro_menu</code>	<code>menu_label</code> , <code>alt_text</code> , <code>description</code>
<code>wpro_menuButtons</code>	<code>display_text</code> , <code>alt_text</code> , <code>image_alt_text</code>
<code>wpro_module</code>	<code>page_title</code> , <code>alt_text</code> , <code>module_label</code>
<code>wpro_special_detail</code>	<code>short_desc</code> , <code>long_desc</code>

Wpro_registry

Use the `getDataValue()` function for translation of registry values. The `wpro_registry` table keys for override labels:

OP module: “`itemTypesTopLabel`”, “`itemTypesLabel0`”, “`itemTypesLabel1`”, “`itemTypesLabel2`”, “`itemTypesLabel3`”, “`itemTypesLabel4`”, “`itemTypesLabel5`”

SYS module: “`itemLayoutLabel`”, “`itemLayoutHeader`”.

Create new keys (use existing codes followed by “-”, followed by language code) with translated values in `wpro_registry` table.

Images

Displayed images with literals will require translation. Refer to the sub directory in the image path for the following images:

add_to_cart_light.gif	copy.gif
move_off.gif	delete.gif
move_on.gif	gobtn.jpg
save.gif	login_button.gif
select.gif	logout_button.gif

Images with the translated literals will be copied into the sub directory (e.g. images/eng/).

Implementation

When these images are referenced, simply add language codes to the end of image path.

Database Tables

The following database tables must be set up for internationalization:

Table	Purpose
Wpro_currency	Holds currency exchange rates. Used in ordering module.
Wpro_data_trans	Holds strings in database translation.
Wpro_error_action	Indicate what action types ("Html", "Email", "Log") to take for the error.
Wpro_error_msg	Holds Javascript error messages and Progress error messages.
Wpro_idx	Holds field names (a comma separated list) of the unique index in database tables. Used in getPUK() function.

Functions Used In Internationalization

getDatalate()

Purpose: To get the translated data value of CSS database fields. Used with getPUK() function.

API: getDatalate (INPUT kTableName AS CHARACTER,
INPUT kFieldName AS CHARACTER,
INPUT klang AS CHARACTER,
INPUT kFieldValue AS CHARACTER,
INPUT kPUK AS CHARACTER)

Return Value: Translated string

Parameters:

Input: kTableName - The name of CSS database table name requires translation.

kFieldName - The name of CSS database field name requires translation.

kLang - Language code for the current session.

kFieldValue - The database field value in this table/field requires translation.

kPUK - Unique index value for the database record requires translation. Function getPUK() is used to obtain the value.

Data Details

This function uses wpro_data_trans to store all data translation. Database Table: wpro_data_trans

Field Name	Description
translated_string	The value in a database field will be translated into this string.
wpro_lang	The language used for translation.
wpro_table	The name of the database table which field value requires translation.
wpro_field	The name of the database field which value requires translation
wpro_table_key	The string of concatenated database field values from field names in wpro_idx.wpro_field_list. The value represents the unique index in a database record requires translation. GetPUK() function will return the value of the field.

putCharSet()

Purpose: This function inserts a <META> tag into Html page between <HEAD> tags. The main purpose is to state what set of characters should be used to display the code within the document.

API: putCharSet().

Return Value: none

Parameters: none

Data Details

Character set is stored in wpro_lang_control.character_set. The value is “iso-8859-1” for English.

getPhraselate()

Purpose: To translate static labels or messages in html pages with embedded parameter(s). These parameters will be replaced with actual values in this function.

```
API: getPhraselate (INPUT kSessionNumber AS
CHARACTER,
INPUT kUserLang AS CHARACTER,
INPUT kString AS CHARACTER,
INPUT kParamVars AS CHARACTER,
INPUT kParamValues AS
CHARACTER)
```

Return Value: Translated string

Parameters:

Input: kSessionNumber - This is the Standard CSS session ID.

kUserLang - Language code for the current session.

kString - The string requires translation.

kParamVars – (optional) A CHR(1) delimited list of parameters, in the form “[@parameter]”, to be replaced by a specified value from the kParamValue list at runtime.

kParamValues – (optional) A CHR(1) delimited list of values directly associated with the above kParamVars. Same number of entries that appear in kParamVars. The value to replace each parameter.

Data Details

All string translations are stored in wpro_string_trans table.

Database Table: wpro_string_trans

Field Name	Description
translated_string	The translated string for the string in trans_string, with the language in wpro_lang. The output string of this function.
trans_string	A string requires translation. This field and wpro_lang make up the primary unique index for the table.
wpro_lang	The language used for the translated string.

getXlate()

Purpose: To translate static labels or messages in html pages. If the input string does not exist in the database, the input string will be returned as the translated string.

API: getXlate (INPUT kString AS CHARACTER).

Return Value: Translated string

Parameters:

Input: kString - The string that requires translation.

Data Details

All string translations are stored in wpro_string_trans table.

Database Table: wpro_string_trans

Field Name	Description
translated_string	The translated string for the string in trans_string, with the language in wpro_lang. The output string of this function.
trans_string	A string requires translation. This field and wpro_lang make up the primary unique index for the table.
wpro_lang	All string translations are stored in wpro_string_trans table

getPUK()

Purpose: To get the unique key values for a given rowid of the database record. Used by getDatelate().

API: getPUK (INPUT kTableName AS CHARACTER,
INPUT rROWID AS ROWID).

Return Value: Unique Key value

Parameters:

Input: kTableName - The name of CSS database table name for the unique key value returned.
rROWID - The row id of the record that will have its unique key value returned.

Data Details

The output string is the key value consist of a concatenated string of table/field values. The database table used for storing unique index field names is wpro_idx.

Database Table: wpro_idx

Field Name	Description
wpro_table	Primary unique index. The name of the database table which unique index field names store in wpro field list field.
wpro_field_list	A database table (wpro_table) unique index field names in comma separated list.

setProcessError()

Purpose: To provide a standard method for the processing of error conditions which might occur in CSS. This function will accept a list of parameters to allow a developer to tailor its functionality

API: setProcessError (INPUT kSessionNumber,
INPUT kErrorNumber,
INPUT kAction,
INPUT kOverride,
INPUT kMessageVars,
INPUT kMessageValues).

Return Value: Error message

Parameters:

Input: kSessionNumber - This is the Standard CSS session ID. It is used to determine the language in use for the current session.

kErrorNumber - Error identifier. An eight-character string (comprised of a two-letter abbreviation followed by a six-digit or a three-letter module abbreviation followed by a five-digit integer) uniquely identifying a specific error message in the wpro_error_msg.error_number field. This function will only look at the Progress error (wpro_error_msg.wpro_type equals "pro").

kAction (optional) - A CHR(1) delimited list of actions to be overridden. Valid actions are any action listed in the corresponding wpro_error_action table.

kOverride (optional) - A CHR(1) delimited list of logical values directly associated with the above list of actions. Valid values are "TRUE" or "FALSE".

kMessageVars (optional) - A CHR(1) delimited list of placeholders, in the form "[@placeholder]", to be replaced by a specified value from the kReplaceValue list at runtime.

kMessageValues (optional) - A CHR(1) delimited list of values directly associated with the above list of replacements to be made.

Data Details

The function will gather all necessary information from two tables in the database (wpro_error_msg, wpro_error_action).

Database Table: wpro_error_action

Field Name	Description
error_number	Primary key. Links one or more records to a specific wpro_error_msg record.
error_action_code	A character string denoting some action for a specified message. There are "html", "log", "email".
error_active	Perform this action? (True/False).
error_desc	A full description of the action being performed

Database Table: wpro_error_msg

Field Name	Description
error_number	A character string in the general form of xx999999 or xxx99999 where the message is used. The first two characters represent the module (i.e. "op" for Order Processing). For the form of xxx99999 is for Ex31, first two characters is module name followed by the "x". The remaining digits are assigned in sequential order beginning with 000001.
wpro_lang	The language currently in use. Like wpro_lang_control.wpro_lang.
error_msg	The actual message to be output by the setProcessError function
error_desc	A full description of the error for providing a more in-depth discussion of the error condition.
wpro_module	The Module name, if applicable, where this message is used.
wpro_type	This field denotes whether the message is a Progress ("pro") or JavaScript ("js") message. Allows for special processing by message type.

getDataValue()

Purpose: To translate wpro_registry table with data override of labels in a Html page. This function rebuilds the key value (wpro_registry.wproKey) to the wpro_registry and passes into getValue() function. All wpro_registry records require translation will have the key value wpro_registry.wproKey followed by “-“, followed by language code (e.g. “itemLayoutHeader-eng”).

API: getDataValue (INPUT kInput AS CHARACTER,
INPUT kLang AS CHARACTER)

Return Value: Translated string

Parameters:

Input: kInput - This is the same input to the function getValue().
Entry one is the key value to wpro_registry, entry two is the module code.

kLang - Language code used in current session.

Data Details

One of the usages of wpro_registry table is to override labels in a Html page. We extended wpro_registry.wproKey value to include language code for the data values that require translation.

APPENDIX D

International Installation

Requirement

1 Database table records

- WPRO_IDX: This table holds the comma separated list of unique index field names of database tables.
- WPRO_TRANS_FIELD: This table holds databases table/field names require translation.
- WPRO_CURRENCY: This table holds the currency unicodes.
- WPRO_ERROR_ACTION, WPRO_ERROR_MSG: These tables hold the event messages for Javascript or Progress and the actions to be taken.

2 Set up a default language.

Default language set up is mandatory for CSS internationalization. Several functions in the system are depending on the default language.

3 Set up registry keys.

- "imageDir": the directory where the images is stored
- "jsmsgDir": the directory where the javascript messages is stored
- "menuDir": the directory where the menu javascript is stored
- "userLangItems": This flag control the item and product category searches. When set to no the users language is not considered. When set to yes it will use the users current language code. The value can also be set to a specific language code that will return items for that code.

4 Set up Email Events.

Installation Procedures for Business Partners

- 1 Go to Language Control Maintenance to set up a new language.
Users can choose to copy database tables for translation from the default language to the new language. Go to Step 2 to dump these records for translation. Javascript message files, Images files and menu Javascript files will be created too.
 - Javascript Message Files: Create `err_msg.js` for the new language from `wpro_error_msg`, places in the sub directory of scripts. Create the sub directory name of language code followed by “_msg” (e.g. `esp_msg`), if it does not exist.

New file/directory:

- `scripts/eng_msg/err_msg.js`
- `scripts/esp_msg/err_msg.js` (Spanish)
- `scripts/pt_msg/err_msg.js` (Portuguese)
- Images Files: `add_to_cart_light.gif`, `move_off.gif`, `move_on.gif`, `save.gif`, `select.gif`, `copy.gif`, `delete.gif`, `gobtn.jpg`, `login_button.gif`, `logout_button.gif`, to sub directory of images. Create the sub directory name of language code (e.g. `esp`), if it does not exist.

New file/directory:

- `images/eng/add_to_cart_light.gif`, etc.
- `images/esp/add_to_cart_light.gif`, etc. (Spanish)
- `images/pt/add_to_cart_light.gif`, etc. (Portuguese)
- Menu Javascript Files: Places in the sub directory of menu. Create the sub directory name of “it_” plus language code.

New file/directory:

- `menu/it_eng.js`
 - `menu/it_esp.js` (Spanish)
- 2 Dump data into text files for translation. (`dump_data.p` see attachment). These tables will be created from Step 1. A log file is created in the same directory that all files are dumped to. The name of the log file is language code plus “_” plus “dump.log”.
 - `WPRO_DATA_TRANS`: Translate the text in fifth column to the new language string into the fifth column.

- WPRO_ERROR_MSG: Translate the text in third column to the new language string into the third column.
 - WPRO_STRING_TRANS: Translate the text in second column to the new language string into the third column. **EXCEPTION: If text in second column prefix with “zz9236#?”, translate the text in third column into the third column.**
 - WPRO_ITEM_TYPES: Translate the text in the first column to the new language string into the first column.
 - WPRO_INFO_WINDOW_DETAIL: Translate the text in the first column to the new language string into the first column, translate the text in the second column to the new language string into the second column.
- 3 Load data into database. (load_data.p see attachment) A log file is created in the same directory that all files are loaded from. The name of the log file is language code plus “_” plus “load.log”.
- 4 Administration. Users and translation adjustments. Maintain database tables through the following:
- Set up users. Select "User Maint" menu, add or modify users.
 - WPRO_DATA_TRANS: Database field value translation. Translation can be done by clicking the “DataLate” icon for the following tables/fields:

Maintenance

Browser Maintenance
 Customer Care
 Email Master Maintenance
 Icon Maintenance
 Menu Maintenance
 Menu Button Maintenance
 System Module Maintenance
 Special Detail Maintenance

Fields for datalate

Browse Title, List of Labels
 Contact Title
 Subject, Body
 Alt Text, Image Alt Text
 Menu Label, Menu Alt Text, Menu Descript.
 Alt Text, Image Alt Text, Display Text
 Menu Label, Alt Text, Page Title
 Short Description, Long Description

- WPRO_STRING_TRANS: String Translation Maintenance. Translate all strings in new language.
- WPRO_ERROR_MSG: Event Message Maintenance. Translate all strings in new language.
- WPRO_CUST_ITEMS: Customer Item Maintenance. Translate all strings in new language.
- WPRO_ITEM_TYPES: Item Types Maintenance. Translate all strings in new language.
- WPRO_FAQ: Faq Maintenance. Translate all strings in new language.

- WPRO_FAQCATEGORY: Faq Category Maintenance. Translate all strings in new language.
 - WPRO_INFO_WINDOW_DETAIL: Message Maintenance. Translate all strings in new language.
 - WPRO_CARRIER_HREF: Carrier Href Maintenance. Translate all strings in new language.
 - WPRO_REGISTRY: All keys end with language code (e.g. "-eng") are used in language translation. (e.g. itemTypesLabel0-eng", "itemTypesLabel0-esp", etc.).
- 5** Replace images with new ones. Copy/replace add_to_cart_light.gif, move_off.gif, move_on.gif, save.gif, select.gif, copy.gif, delete.gif, gobtn.jpg, login_button.gif, logout_button.gif into images/esp or images/pt directory.

APPENDIX E

CSS Tables

Table Name	Description
wpro_active	This table holds a share lock to count the number of active web agents.
wpro_activity_code	This table holds the services activity code used in the help desk
wpro_addr	The CSS address table for temporary address storage during order or RMA processing.
wpro_admin_xref	The Site Administrator cross-reference table for external administration.
wpro_browse_mstr	This table holds the definition for CSS browses.
wpro_call_class	This table holds the valid issues status for customers to enter.
wpro_call_header	This table holds the customer call information.
wpro_call_line	This table holds the customer call detail information
wpro_carrier_href	This table holds the carrier HREF link information for order tracking.
wpro_cc_control	This table contains the control values for the credit card software.
wpro_cc_det	This table holds credit card transaction log detail.
wpro_cc_log	This table holds the credit card transaction log
wpro_cimpool	This table contains the dynamic temp-table definitions for CIM Field groupings or pools.
wpro_contract	This table contains the customer contract information.
wpro_control	This table contains the CSS system control information.
wpro_currency	This table holds the information required by CSS for currency support.
wpro_cust	This table contains the CSS customer information.
wpro_customer_care	This table contains contact information for users.
wpro_cust_item	This table contains the CSS customer item information.
wpro_data_trans	This table contains the information required for data translation.
wpro_debug	This table contains the CSS debugger log.
wpro_dio_update	This table contains the CSS record state information.

wpro_doc	The table contains the CSS developer doc for functions and their API.
wpro_dtmap	This table hold the definitions for dynamic tables
wpro_emev_detail	This table contains the email detail information.
wpro_emev_journal	This table contains the CSS email log information
wpro_emev_master	This table contains the definition of CSS email events.
wpro_enhancement	This table contains the list of product enhancements.
wpro_enhancement_detail	This table contains the detail enhancement information.
wpro_error_action	This table contains what action should be taken when an error condition occurs.
wpro_error_msg	This table contains the definitions of CSS errors.
wpro_faq	This table contains the list of questions and answers to frequently ask questions.
wpro_faqCategory	This table contains the list of valid FAQ categories.
wpro_help	Associates CSS HTML pages with help pages and anchors for a customizable help system.
wpro_icons	This table contains information in the icons used in "Order Entry".
wpro_idx	This table contains the definitions for the unique index of tables maintained in CSS for translations.
wpro_info_window_detail	This table contains data to be displayed in the information frame at launch point.
wpro_info_window_master	This table defines the layout type for the launch point page.
wpro_item_types	This table contains parent / child item type relationships.
wpro_knowledge	This table contains the knowledge base for the CSS support system,
wpro_lang_control	This table contains the language master information.
wpro_layout_master	This table contains IDs and descriptions of the different layout schemes for launch points.
wpro_menu	This table contains the definitions and relationship of CSS functions on the CSS menu.

wpro_menuButtons	This table contains the objects available on the CSS menu bar.
wpro_menu_detail	Menu item detail information used by the launch point menu.
wpro_menu_master	Table used for holding top level menu items for the launch point menu.
wpro_module	This table contains the definitions for all CSS modules (HTML pages).
wpro_module_js	This table contains the list of all JavaScript files required for each CSS module.
wpro_msg	This table contains the default message information for CSS.
wpro_msg_det	This table contains the detail information for CSS messages.
wpro_order_archive	Table for storing archived data from CSS orders.
wpro_order_archiveC	Table for storing archived data from CSS order line for configuration.
wpro_order_archiveL	Table for storing archived data from CSS order lines.
wpro_order_control	This table contains the default order control information.
wpro_order_header	This table contains the basic information about orders in CSS.
wpro_order_line	This table contains the line detail information about orders in CSS.
wpro_order_line_comp	This table contains the line config detail information about orders in wpro_order_line_comp.
wpro_problem	This table contains the problem descriptions.
wpro_problem_line	This table contains the problem description details.
wpro_registration	This table contains the CSS registration information
wpro_Registry	This table contains the CSS registry information.
wpro_rma_control	This table contains the system control information for RMA's.
wpro_rma_header	This table contains the base information about RMA's.
wpro_rma_line	This table contains the RMA detail information.
wpro_rpt_mstr	This table contains the base report definitions for generated reports.
wpro_rpt_params	This file holds the report input parameters for report generation.
wpro_security_groups	Define the menu security groups available in CSS

wpro_session	This table contains the base CSS session information.
wpro_session_value	This table contains the variable session information.
wpro_sms_security	This table contains additional security profile information for the SMS.
wpro_solution	This table contains the base information for the solution records.
wpro_special_detail	This table contains the detail information about item specials.
wpro_special_master	This table contains the base information about item specials.
wpro_special_reg	Users who register to be notified of promotions / specials are stored here.
wpro_status_code	This contains the list of valid status codes.
wpro_string_trans	This table contains the CSS character strings and their translation
wpro_super	This table contains the list of CSS persistent modules.
wpro_support_level	This table contains the list of valid support call levels.
wpro_table_return	This table contains the data for custom browse rows.
wpro_tech	This table contains a list of valid support technicians.
wpro_TransactionLog	This table contains the CSS transaction log.
wpro_trans_field	This table contains a list of tables/ fields that are translatable.
wpro_user_ccard	This table contains user credit card information.
wpro_user_detail	This list contains the relationship of CSS user to the ERP system.
wpro_user_master	This table contains the list of valid CSS users.
wpro_version	This table contains the valid list of CSS releases.

APPENDIX F

Debug Module

Overview

The proposed CSS Debug module is a response to the need to know what code was executed in order to generate a specific Web page through CSS. This module will also identify which functions or internal procedures were called and what Super Procedure are they defined in. It will specify the different APIs and their syntax and what information was passed between them. The developer will be able to view the query string that produced a list of data records. The debug module will be included in the base release but can be turned on or off and to some degree customizable per developer.

Stack Trace

This is the first phase of the Debug module. When implemented, it will capture data within each API and store it in a debugger repository inside the CSS database. That data will then be read and displayed at the end of each Web page for the developer to review. The results will show the developer the order that the objects were accessed, what the calling object was, and where the objects reside in the Super Procedures.

The information will appear similar to the following layout:

TransID:	14056	Date:	9/15/01		
Called Object (API)		Parent Object (API)		Calling Object (API)	Elapsed Time
getSomeInfo1		lib_sys_superex2.p		login.r	15
setSomeData		lib_sys_superui1.p		getSomeInfo1	10
getSomeMoreInfo		lib_sys_superex2.p		getSomeInfo1	15
putHTMLfunc1		lib_sys_superui2.p		login.r	20

Stack Trace Features

Each called object will be displayed as a link. When clicked, it will display the documentation on that function or procedure. The displayed information will show the purpose of the object, parameters, and return value if it is a function.

The developer can also select how much information is to be displayed.

General / Specific Messages

In the second or later phase of the Debug module, a feature will be added to allow general messages as well as error messages to be made accessible from the Debug log screen. At the time of this writing, this feature needs more discussion as to how it will be developed and implemented.

{&EXDEBUG}

The preprocessor EXDEBUG will equate to a function call. The function will be setExDebug.

setExDebug

This function will look at the user's session information and determine whether or not debugging is turned on or not. The function that will indicate whether or not debugging is turned on is putExDebugSwitch.

If it indicates a positive value, then setExDebug will gather the necessary information and create debug records for that user's transactions. When the web transaction is finished, a footer will be appended to the output web page which will display the stack trace information. The function that will build the HTML table and populate it with the stack trace info is putExDebug.

APPENDIX G

Standards

File Names

Directory and Filename Rules:

- 1 All lower characters
- 2 2-3 character module directory names
- 3 All file names will start with module directory prefix
- 4 Use only “_” (underscores) and not “-“ hyphens.

Convention syntax:

module_dir/mod_func_unique-name(n).ext

<i>module_dir:</i>	module directory name
<i>mod:</i>	module reference
<i>func:</i>	type of procedure (rpt, rpe, brw, inq, etc)
<i>unique-name:</i>	verbose file name descriptor
<i>n:</i>	sequence number of procedure if needed
<i>ext:</i>	extension (html, p, etc)

Old name example:

op/rpt/op_rpe_customer_credit_entry.html

New file name examples:

op/op_rpe_custcredit.html

op/op_rpt_custcredit.html

op/op_mnt_customer2.p

op/op_mnt_customer3.p

Temp-Table Standards

All temp-tables will adhere to the following:

- All temp-table names will begin with the “**tt_**” prefix. Following the prefix will be the table name that the temp-table is related to, i.e., *tt_customer*.
- Every temp-table will contain the following two fields:

tbSrtOrder	AS INTEGER
tbRowId	AS ROWID

Defined Variables

Variables will indicate by their defined name, what type of data-type they are storing. The name will begin with one of the following codes:

c	Decimal
i	Integer
d	Date
k	Character
l	Logical
h	Handle
r	Rowid
rd	Recid
rw	Raw

Following the variable identity code will be the actual variable name in Hungarian notation.

Internal Procedures & Functions

All internal procedures and functions will adhere to the following guidelines.

Naming:

Procedure and Function names will begin with a lower case word from the following list:

- get** – defines a procedure or function that will return a value
- set** – defines a procedure or function that will update a db record
- put** – defines a procedure or function that will output HTML code
- val** – defines a procedure or function that will apply some business rules
- gen** – defines a procedure or function that will generate a screen
- wrap** – is a wrapper that converts a procedure to a function

Following this word will be an “InterCap” or Hungarian style naming convention.

When defining a function or procedure, define all input parameters first, then output parameters and lastly input-output parameters. All functions and procedures should have a comment body preceding the definition. *See section on comment body for details.*

Comment Body

The following comment template will be used for all internal procedures and user-defined functions:

```

    /** CSS Documentation
    ****
    Procedure/Function Name:
    Purpose:
    Return Value:
    Parameters:
        Input:  none
        Output: none
        Input/Output: none
    **** End CSS Documentation
  
```

Procedure/Function Name: *Enter the procedure or function name as defined. All functions should include the following parenthesis.*

Purpose: *If the purpose is known or specified it should be as short as possible but as long as is necessary to explain the contained behavior.*

Note the purpose statement must not span wider than the asterisk lines that surround the comment template.

Return Value: *Specify the defined return value for all functions. Leave blank for procedures.*

Parameters:

Input: *Specify each parameter on a separate line. Each one must specify its data type.*

Output: *If there are no input, output or input-output parameters, specify “none”.*

Input/Output: **same as above**

Internal Procedure Example:

```

/** CSS Documentation *****/
  Procedure/Function Name:  pageNavigation
    Purpose:
  Return Value:  None
    Parameters:
      Input:  curRecCnt AS CHARACTER
             repPerSet AS CHARACTER
             totRecCnt AS CHARACTER
      Output: stPage    AS INTEGER
             sePage    AS INTEGER

```

Input/Output: None

```

**** End CSS Documentation *****/
PROCEDURE pageNavigation:
  DEFINE INPUT PARAMETER curRecCnt AS CHARACTER    NO-UNDO.
  DEFINE INPUT PARAMETER recPerSet AS CHARACTER    NO-UNDO.
  DEFINE INPUT PARAMETER totRecCnt AS CHARACTER    NO-UNDO.
  DEFINE OUTPUT PARAMETER stPage AS INTEGER        NO-UNDO.
  DEFINE OUTPUT PARAMETER sePage AS INTEGER        NO-UNDO.

  DEFINE VARIABLE C1 AS INTEGER NO-UNDO.
  DEFINE VARIABLE C2 AS INTEGER NO-UNDO.

  {&EXDEBUG}

```

Function Example:

```

/** CSS Documentation *****/
Procedure/Function Name: useCustomer()
    Purpose:
    Return Value: CHARACTER
    Parameters:
        Input: llCust AS CHARACTER
        Output: none
    Input/Output: none
**** End CSS Documentation *****/
FUNCTION useCustomer RETURNS CHARACTER (INPUT llCust AS CHARACTER):
    DEFINE VARIABLE useCust AS CHARACTER NO-UNDO.

    {&EXDEBUG}

    IF getValue("searchList,op") = "customer" THEN
    DO:
        IF getValue("useMasterCustomer,op") = "yes" then
            ASSIGN useCust = getMasterCust(llCust).
        ELSE
            ASSIGN useCust = llCust.
        END.
    ELSE
        ASSIGN useCust = "".
    RETURN useCust.
END.

```

{&EXDEBUG} Preprocessor

Every internal procedure and user defined function will have the {&EXDEBUG} preprocessor inserted on a line by itself proceeding all parameters and definitions and preceding the remaining lines of code in the procedure or function definition. *See previous examples.*