Release 1923

Service Request 82349
Web New Hire

Detail Design

May 14 2010
Revised June 7, 2010

Prepared by Maxine Gerber
Peter Parker
Leo Herrick
Steve Hunter
Rajeev Gupta

Information Resources & Communications
Office of the President
University of California
# Table of Contents

I. Introduction ........................................................................................................................................................................ 1  
1. Service Request 82349 ...................................................................................................................................................... 1  
II. Overview of PPS Modifications ........................................................................................................................................ 1  
III. PPS Web Application Supported Technologies .............................................................................................................. 1  
IV. General System Design ................................................................................................................................................... 2  
V. Mainframe Design ................................................................................................................................................................. 2  
   1. CICS Web Services .......................................................................................................................................................... 2  
   2. RACF Security ................................................................................................................................................................. 3  
      2.1. PS020 ........................................................................................................................................................................... 4  
      2.2. Security Prefix .......................................................................................................................................................... 4  
      2.3. Web Hire Security .................................................................................................................................................. 4  
      2.4. CICS Function Security .......................................................................................................................................... 4  
   3. Cobol Programs ................................................................................................................................................................. 4  
      3.1. PPAPEUFE ................................................................................................................................................................. 4  
      3.2. PPCANHIR (new) ................................................................................................................................................... 4  
      3.3. PDELETS .................................................................................................................................................................. 5  
      3.4. PLCNUPD ................................................................................................................................................................. 5  
      3.5. PPWPLT ................................................................................................................................................................. 5  
      3.6. PPWWEDB (new) ................................................................................................................................................... 5  
      3.7. UCROUTER ............................................................................................................................................................ 5  
      3.8. UCWABND ............................................................................................................................................................... 5  
      3.9. UCWECOM ............................................................................................................................................................. 5  
      3.10. UCOMLR ............................................................................................................................................................... 6  
   4. Web Service Cobol Programs .......................................................................................................................................... 6  
      4.1. PS006 (new) ............................................................................................................................................................. 6  
      4.2. PS007 (new) ............................................................................................................................................................. 6  
      4.3. PS008 (new) ............................................................................................................................................................. 6  
      4.4. PS008D (new) ......................................................................................................................................................... 6  
      4.5. PS012 (new) ........................................................................................................................................................... 6  
      4.6. PS013 (new) ........................................................................................................................................................... 7  
      4.7. PS015 (new) ........................................................................................................................................................... 7  
      4.8. PS017 (new) ........................................................................................................................................................... 7  
      4.9. PS018 (new) ........................................................................................................................................................... 7  
      4.10. PS021 (new) ......................................................................................................................................................... 7  
      4.11. PS022 (new) ......................................................................................................................................................... 7  
      4.12. PS023 (new) ......................................................................................................................................................... 7  
      4.13. PS024 (new) ......................................................................................................................................................... 8  
      4.14. PS025 (new) ......................................................................................................................................................... 8  
      4.15. PS026 (new) ......................................................................................................................................................... 8  
      4.16. PS027 (new) ......................................................................................................................................................... 8  
   5. Copy Members ................................................................................................................................................................. 8  
      5.1. CPPDFKEY (new) .................................................................................................................................................... 8  
      5.2. CPPDFLT ................................................................................................................................................................. 8  
      5.3. CPPDTOKN ............................................................................................................................................................. 8  
      5.4. CPWSFKFMK (new) ............................................................................................................................................ 8  
      5.5. CPSSTOKK ............................................................................................................................................................. 8  
      5.6. CPWSWEBS .......................................................................................................................................................... 9  
      5.7. CPWWEDB (new) ............................................................................................................................................ 9  
   6. DDL Members ................................................................................................................................................................. 9  
      6.1. PPSUS Table (new) ............................................................................................................................................. 9  
      6.2. PPSUSX Table (new) ....................................................................................................................................... 9  
      6.3. PPPLTA Table (new) ........................................................................................................................................ 9  
      6.4. Indexes to Improve Performance ....................................................................................................................... 10
VI. Web Design ......................................................................................................................................................... 13
1. Technology Stack ............................................................................................................................................... 13
2. Security and Authentication .......................................................................................................................... 15
3. Screens ............................................................................................................................................................ 15
   3.1. Employee Identification ......................................................................................................................... 15
   3.2. Personal Information .............................................................................................................................. 15
   3.3. Addresses and Disclosures .................................................................................................................... 16
   3.4. Citizenship and Taxes ............................................................................................................................ 16
   3.5. Employment Information ....................................................................................................................... 17
   3.6. Appointments and Distributions ........................................................................................................ 17
   3.7. Education ............................................................................................................................................... 18
   3.8. Pay Disposition ...................................................................................................................................... 19
   3.9. Summary ............................................................................................................................................... 19
   3.10. Review Messages and Update ............................................................................................................ 19
   3.11. Post-Authorization Notice ................................................................................................................ 19
   3.12. IDOC .................................................................................................................................................... 20
4. Model .................................................................................................................................................................. 20
   4.1. Employee Object .................................................................................................................................. 20
   4.2. Data Elements ..................................................................................................................................... 20
5. View .................................................................................................................................................................. 23
   5.1. Tiles ......................................................................................................................................................... 23
   5.2. Tiles Schematic .................................................................................................................................... 24
   5.3. Controller ............................................................................................................................................... 24
   5.4. Struts Configuration ............................................................................................................................... 24
   5.5. Struts Themes ....................................................................................................................................... 25
6. Pivot Fields ........................................................................................................................................................ 25
   6.1. Pivot Field Inventory .............................................................................................................................. 26
7. Range Value Edits .......................................................................................................................................... 26
8. Navbar Invocation ................................................................................................................................========== 26
9. Next/Prev Invocation ..................................................................................................................................... 27
10. Consistency edits .......................................................................................................................................... 27
11. PAN .............................................................................................................................................................. 27
   11.1. Add recipient ....................................................................................................................................... 27
   11.2. View PAN Notice ................................................................................................................................. 28
12. IDOC ............................................................................................................................................................. 28
13. Templates ....................................................................................................................................................... 29
   13.1. Using a Template .................................................................................................................................. 29
   13.2. Saving a Template ............................................................................................................................... 30
14. Application Properties .................................................................................................................................. 30
   14.1. Code Translations ............................................................................................................................... 30
   14.2. Externalized Properties File .............................................................................................................. 30
   14.3. Consistency Edit Severity Levels ....................................................................................................... 31
   14.4. Campus Customizations .................................................................................................................... 31
   14.5. Drop Down Code Filtering .............................................................................................................. 31
   14.6. Allow Add PAN Reviewer by Email .................................................................................................. 32
   14.7. Visible Foreign Visa Data Elements ................................................................................................. 32
   14.8. Display Registered Units Data Element .......................................................................................... 32
I. Introduction

1. Service Request 82349

Service Request 82349 asks for the creation of a web interface to the EDB update system of PPS. This will be used as an alternative to the existing CICS interface.

Service Request 82349 provides the following background information:

Developing a Web EDB Update front end to the mainframe EDB system provides the opportunity to modernize and enhance the view or interface between users and the system by developing a new look while leveraging the existing technically-complex EDB mainframe system.

The development team plans to take a Service Oriented Architecture (SOA) approach to this project; creating a number of web services on the mainframe system that will expose existing PPS architecture to the web. The goal is to leave existing system logic as is, and to create an external web application that will use the web services to pass information and to receive information from PPS.

II. Overview of PPS Modifications

This release will create a new web interface for PPS users, allowing them to add newly hired employee data to the EDB. Existing COBOL PPS programs and existing system logic will largely be left unmodified. The few changes made to existing PPS programs will only be done so as to make them compatible with new set of web service programs.

**Important Note:** The web new hire application will require that automatic ID generation be on for the campus. System parameter #64 must be turned on (value = 1).

A set of web service programs will be created to perform all queries and updates between the web interface and PPS. The web interface will not directly query DB2 or directly execute existing PPS COBOL programs. All interaction with the system will be through these web services. Separate services will be created for such actions as running range/value edits, assigning an employee ID, and adding PAN recipients.

Two new tables will be created to support the proposed “Suspend” and “Templates” features in the application. These features allow users to a) save partially completed transactions, and b) save certain sets of data element name|value pairs in the system for future use. This saved information will be keyed by username and saved in DB2.

Modifications will be made to PPS Web Main Menu to allow users to enter the New Hire application. A new link will be added named "EDB Update System - New Hire".

III. PPS Web Application Supported Technologies

**Supported browsers:**
- IE 7.0, 8.0
- Firefox, 2.0, 3.6

**EDBUpdate New Hire use the following technologies**
- Struts 2 Release 2.1.6
- Struts Tiles 2.1.2
- Struts 2 OGNL 2.6.11
- Spring Framework 2.5.3
• xwork 2.1.2
• Jaspereports 3.7.0
• Itext 2.1.0
• Textile 2.2
• Various Apache commons libraries
• Javascript Libraries:
  • jQuery 1.3.2
  • Jquery-ui 1.7.2
  • Autocomplete 1.0.2 - jQuery plugin
  • Autotab 1.1b - jQuery plugin
  • Bestupper 1.0 – jQuery plugin
  • jqGrid 3.5.2 - jQuery plugin
  • jQuery Form 2.28 - jQuery plugin
  • Masked Input 1.0 – jQuery plugin

PPSWebServices uses the following technologies
• Sun Metro 1.5

IV. General System Design

The new user interface is architected in the following manner.

A Java application, deployed on a WebSphere application server, will serve HTML forms to the end user. The end user will use these forms to input employee information and submit it back to the application. The Java application will gather user-submitted data and use the SOAP specifications to exchange information with Web Service programs on the PPS Mainframe system. Web services will interact with existing PPS programs to update or retrieve PPS data, and use SOAP to send that information back to the web application.

The application will also serve a set of JavaScript files to the client’s browser, which will be responsible for running special interactions on the front end.

V. Mainframe Design

1. CICS Web Services

Each web service is either channel-based or commarea-based.

A channel-based web service is developed starting with a copylib member which defines the input and output structure for the service. This copylib member is named the same as the web service (eg. PS006). It is never copied into a program, but is imported into RADz to generate the wsdl. After the wsdl has been generated and modified as needed, RADz uses the wsdl to generate two copylibs to be used in the service program, one for the input structure and one for the output structure. These are named PSxxxO01 and PSxxxI01 where PSxxx is the service name.

A commarea-based web service is developed starting with a copylib member which defines the input and output structure for the service. This copylib member is copied into the service program. It is also used by RADz to generate the wsdl.

The following is a list of the web services components in this release. For more detail on each service, see the Cobol program description.

<table>
<thead>
<tr>
<th>Service</th>
<th>Channel or Commarea</th>
<th>Cobol Program(s)</th>
<th>Copylib Member(s)</th>
<th>Wsdl</th>
<th>Description</th>
</tr>
</thead>
</table>

Page 2
<table>
<thead>
<tr>
<th>Channel</th>
<th>PS006</th>
<th>PS006</th>
<th>PS006I01</th>
<th>PS006O01</th>
<th>PS006.wsdl</th>
<th>Consistency Edits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td>PS007</td>
<td>PS007</td>
<td>PS007I01</td>
<td>PS007O01</td>
<td>PS007.wsdl</td>
<td>Save or retrieve a “template” of entered data created by the user</td>
</tr>
<tr>
<td>Commarea</td>
<td>PS008</td>
<td>PS008</td>
<td>PS008I01</td>
<td>PS008O01</td>
<td>PS008.wsdl</td>
<td>Commit an EDB update</td>
</tr>
<tr>
<td>Channel</td>
<td>PS012</td>
<td>PS012</td>
<td>PS012I01</td>
<td>PS012O01</td>
<td>PS012.wsdl</td>
<td>Employee ID assignment</td>
</tr>
<tr>
<td>Channel</td>
<td>PS013</td>
<td>PS013</td>
<td>PS013I01</td>
<td>PS013O01</td>
<td>PS013.wsdl</td>
<td>Code translations to be cached at web server startup.</td>
</tr>
<tr>
<td>Channel</td>
<td>PS015</td>
<td>PS015</td>
<td>PS015I01</td>
<td>PS015O01</td>
<td>PS015.wsdl</td>
<td>Value and range edits</td>
</tr>
<tr>
<td>Channel</td>
<td>PS017</td>
<td>PS017</td>
<td>PS017I01</td>
<td>PS017O01</td>
<td>PS017.wsdl</td>
<td>Format an employee name from its component parts</td>
</tr>
<tr>
<td>Channel</td>
<td>PS018</td>
<td>PS018</td>
<td>PS018I01</td>
<td>PS018O01</td>
<td>PS018.wsdl</td>
<td>Return all PAN data for an update in progress</td>
</tr>
<tr>
<td>Channel</td>
<td>PS020</td>
<td>PS020</td>
<td>PS020I01</td>
<td>PS020O01</td>
<td>PS020.wsdl</td>
<td>Check ARSM</td>
</tr>
<tr>
<td>Channel</td>
<td>PS021</td>
<td>PS021</td>
<td>PS021I01</td>
<td>PS021O01</td>
<td>PS021.wsdl</td>
<td>Update PAN data for an update in progress</td>
</tr>
<tr>
<td>Channel</td>
<td>PS022</td>
<td>PS022</td>
<td>PS022I01</td>
<td>PS022O01</td>
<td>PS022.wsdl</td>
<td>Return an employee document (IDOC)</td>
</tr>
<tr>
<td>Channel</td>
<td>PS023</td>
<td>PS023</td>
<td>PS023I01</td>
<td>PS023O01</td>
<td>PS023.wsdl</td>
<td>Cancel a pending new hire process</td>
</tr>
<tr>
<td>Channel</td>
<td>PS024</td>
<td>PS024</td>
<td>PS024I01</td>
<td>PS024O01</td>
<td>PS024.wsdl</td>
<td>Return a list of suspended hires</td>
</tr>
<tr>
<td>Channel</td>
<td>PS025</td>
<td>PS025</td>
<td>PS025I01</td>
<td>PS025O01</td>
<td>PS025.wsdl</td>
<td>Save or retrieve a suspended hire</td>
</tr>
<tr>
<td>Channel</td>
<td>PS026</td>
<td>PS026</td>
<td>PS026I01</td>
<td>PS026O01</td>
<td>PS026.wsdl</td>
<td>Return system parameter values for requested parameter numbers.</td>
</tr>
<tr>
<td>Channel</td>
<td>PS027</td>
<td>PS027</td>
<td>PS027I01</td>
<td>PS027O01</td>
<td>PS027.wsdl</td>
<td>PAN directory browse</td>
</tr>
</tbody>
</table>

### 2. RACF Security

The Web new hire application uses RACF $PPSFUNC functionality to ensure that an end user is authorized to access the web new hire application, in the same way that it’s used for other web application access.
2.1. **PS020**

Web service PS020 is invoked by the web applications to check $PPSFUNC authorization for specific resources. It calls base PPS program UCFNAUTH.

2.2. **Security Prefix**

Each resource in the $PPSFUNC RACF class has a prefix to define the scope of the resource. For example, for CICS function codes, the CICS region name is the prefix. For the new hire web application, the prefix can be determined locally, and must be stored in the securityprefix property in EDBUpdate.properties file. This value is passed to web service PS020.

2.3. **Web Hire Security**

The $PPSFUNC resource for the web hire application is WHIR. This can be confusing because WHIR is also the name of the bundle used within PPS for the web new hire process. WHIR must be set up as a $PPSFUNC resource with the security prefix defined in the securityprefix property in EDBUpdate.properties file.

2.4. **CICS Function Security**

The CICS web services used by the web new hire application use PPS CICS function codes. The authorization for these is checked during web service calls by UCRouter (and not by the web application).

Two PPS functions are used for new hire:

- WHIR is defined as a bundle. In the UC0CFN table, field CFN_BUN_APPL_DATA is set to “H” defining it as a new hire type of bundle. There is a single function code defined within the bundle, WEDB.
- WEDB is a “generic” EDB update function which can be used for updating any group of data elements.

UCRouter does not check authorization for bundle functions, only for the detail functions within the bundle. Therefore, CICS function authorization only for WEDB must be set up with the usual $PPSFUNC security, using the CICS region name as the prefix.

Note that UCRouter isn’t invoked by web services until the web application user is ready to invoke consistency edits and/or commit the update. Therefore, the function code authorization is not done until that point.

3. **Cobol Programs**

3.1. **PPAPEUFE**

PPAPEUFE is the final edit application processor for the EDB Entry/Update subsystem. It will be changed to suppress the transfer to the ECON screen when a web service is driving the transaction.

3.2. **PPCANHIR (new)**

PPCANHIR will be a called routine used to cancel a new hire in progress. It will be called from programs PS023 and PS025. The program will expect external CPWSXIID-INTERFACE to be present and populated. The program will perform the following functions:

- Delete any pending PAN records from UC0VZPNX_PNX for the current user ID and employee ID.
- Call program UCIIDASN to delete the pending employee ID.
- Call program PPDELTS to delete any temporary storage queues associated with the current session (token).
3.3. **PPDELTS**

PPDELTS deletes all PPS temporary storage queues associated with a given token. It will be modified to also delete pending PAN data and temporary storage queues associated with the web new hire process.

3.4. **PPLCNUPD**

PPLCNUPD drives the update of the PPPLCN table. This program will be modified so that the apply change function for data element 0711 uses data from the “new value” input instead of from the “occurrence key” to populate the license array. This will allow web services to update an existing row in PPPLCN with a different value for element 0711. PPPRGPEN (new)

PPPRGPEN will clean up pending PANs, IDs and expired suspense data. It will be called from PPWPLT at CICS startup only.

3.5. **PPWPLT**

PPWPLT runs at CICS startup to start automatic PPS transactions. Code will be added to start program PPPRGPEN via transaction T011.

3.6. **PPWWEDB (new)**

PPWWEDB will be the "detail screen processor" for the WEDB function. This will function similarly to other EDB update screen processors, except that there will be no BMS map associated with it. Instead, data will be transferred to and from this program through (new) external CPWWEDB which will contain an array of data element numbers and values. This external will be populated by PS006, the consistency edit web service program.

The program will read the XIID temporary storage queue for the token (created when the ID was assigned). If the queue doesn't exist, it's an error. This is done here so that it will be cached by UCROUTER and available for the final update.

PPWWEDB will call PPPVREDO to perform value and range editing. Although there's a separate web service for this editing, it is necessary to perform it again to make sure the data is clean before caching it and starting consistency edits.

If value and range editing is successful, program PPEDBUPD will be called to populate EDB externals. These externals are automatically cached in temporary storage by UCROUTER.

3.7. **UCROUTER**

UCROUTER is the main driver for all PPS CICS processes. It will be been modified to support web service consistency edit navigation as follows:

- If UCROUTER was invoked from service PS006, it will exit after returning from the “FINAL-EDIT” call.
- If UCROUTER was invoked from service PS006, allow PAN processing to proceed. For other web services, PAN processing is bypassed.

3.8. **UCWABND**

UCWABND is the abend handling program for the online applications system. It will be modified to issue a soap fault when processing abends from within web services.

3.9. **UCWECOM**

UCWECOM is the screen processor for the ECOM function (PAN Comments Entry). It will be modified to bypass the code which updates the PNX table when being called in a web service context.
3.10. UC0MLR

UC0MLR is a batch program sends a summary of recently generated PANs to each PAN recipient. It will be modified to exclude PANs in pending status for the recipient’s summary.

4. Web Service Cobol Programs

4.1. PS006 (new)

PS006 will be the driver for the PS006 web service which handles consistency editing for web new hire. It will invoke the consistency edits within PPS via calls to UCROUTER which mimic CICS functionality. It will take as input a list of all data elements to be updated and passes these to a generic "screen handler" program (via UCROUTER).

There will be two modes which are indicated by a request code in the input. The first mode will process consistency edits and return the list of messages regardless of whether or not there are consistency edit errors. In the second mode, if there are no "fatal" consistency edit errors, processing will proceed to the point of PAN interaction, caching all input data in CICS for eventual update. At this point, no additional data may be input. Users may update the PAN and commit the update, or cancel it.

CONSISTENCY EDIT

4.2. PS007 (new)

PS007 will be the driver for the PS007 web service which processes new hire template data. It will save up to 50 data element number/value pairs which can be retrieved to fill in data entry fields. Each “template” will be stored in DB2 table PPPTLA keyed on user id, template type (defined by the requesting Web application), and template name (up to 25 characters, entered by the end user). Templates may be saved, re-saved (updated), or deleted. PS007 also will return a list of all template names for a given user / template type.

4.3. PS008 (new)

PS008 will be the driver for the PS008 web service to commit an EDB update which is already in progress. All consistency edits must have already been completed without fatal errors. The updating will be done from data already cached in CICS so no input data is used for this service.

4.4. PS008D (new)

PS008D will be the the RAD-generated Converter Driver program which will be is invoked by web service PS008. It will parse the XML passed from the web application and then call PS008 to process the data. On returning from PS008, the request output will be translated back to XML for return to the web application.

4.5. PS012 (new)

PS012 will be the driver for web service PS012, the ID assignment service. It will call the appropriate PPS modules to assign a new employee ID, or to use an existing IID system ID for a new employee.

When assigning a new ID, if there are matches on entered data, the service will return an array of possible matches, otherwise, it will return the new ID.
This service will perform some of the logic contained in existing programs PPWEID and PPWIDBS. It will call UCIDASN (and/or UCID100) to return matches, if any, or to assign and return a new (pending) ID. When returning a list of matches, it will check to see if any of the IDs already exist on the EDB and set the appropriate indicator. These cannot be used for a new hire, but will be displayed so users can see if they really should be doing a re-hire.

**Note:**

When a new ID is assigned, but before the hire is committed a row is added to UC0IDB with status=pending. After the hire is committed, the status is changed to active, and a row is added to UC0IXB with system name = 'PP'.

When an existing non-EDB ID is assigned, the IID database is not updated until the hire is committed, at which time the UC0IXB row is added. This means that there's nothing "pending" in IID for this case.

### 4.6. PS013 (new)

PS013 will be the driver for the PS013 web service which will return code values and code translations for requested data elements.

Two request options are supported.
- Return all translations
- Return translations for requested data elements

### 4.7. PS015 (new)

PS015 will be the driver for web service PS015 which will perform value range edits for an array of passed data elements by calling PPPVREDO. It will take as input a list of all data elements to be edited and will return a status code for each.

### 4.8. PS017 (new)

PS017 will be the driver for web service PS018 which will reformat the employee name. It will transform employee name from 1 to 3 parts or from 3 to 1. PPNAMGEN will be called by PS017. In “1 to 3 mode”, the string ‘LLLLLLLLLLL, FFFF M’ is returned as ‘LLLLLLLLLLL’, ‘FFFF’, ‘M’ for last name, first name, and middle name respectively. In “3 to 1 mode”, ‘LLLLLLLLLLL’, ‘FFFF’, ‘M’ for last name, first name, and middle name respectively is return as ‘LLLLLLLLLLL, FFFFF M’

### 4.9. PS018 (new)

PS018 will be the driver for web service PS018 which will return all PAN detail data for a currently active update for the current user.

### 4.10. PS021 (new)

PS021 will be the driver for web service PS018 which will update PAN data for the currently active update process.

### 4.11. PS022 (new)

PS022 will be the driver for web service PS022 which will generate an employee document (IDOC). It will generate all valid versions of the document for the selected employee, ie. staff, academic and student.

### 4.12. PS023 (new)

PS023 will be the driver for web service PS023 which will cancel an active new hire process.
4.13. **PS024 (new)**

PS024 will be the driver for web service PS024 which will return a list of suspended hires that were created by the web hire user.


PS025 will be the driver for web service PS025 which will store and retrieve suspended new hire data. It will provide a facility to save, get, or delete a large binary object (LOB) suitable for saving a Java object that encapsulates the entered data and state of a New Hire. The LOB is saved in DB2 table PPPSUS keyed on the signed on user’s id and the employee id. PS025 will process BLOB’s up to 250K long, returning SOAP Fault if the input is too long.

4.15. **PS026 (new)**

PS026 will be the driver for web service PS026 which provides system parameter data to the web new hire application. It will return system parameter data for the parameter numbers passed in the input array. It will be invoked by the web server at server startup, and once per day while the server is running.

4.16. **PS027 (new)**

PS027 will be the driver for the PS027 which will provide PAN directory data to the web new hire application. It will browse the PAN directory table, UC0DIR on Name.

5. **Copy Members**

For web service copylib member, see the section labeled “CICS Web Services”

5.1. **CPPDFKEY (new)**

CPPDFKEY will contain procedure division code to set the EIBAID function key with the correct value based on the "logical key" value passed to it in working storage. It will search the UCROUTWA-LOGIC-AID-FUNC-ARRAY for the logical key and if a match is found, move the function key value to EIBAID. If no match is found, additional attempts may be made to search for a related logical key. Finally, if no match is found, EIBAID will be set to the enter key. Working storage copylib member CPWSFKEY will be required.

5.2. **CPPDFLT**

CPPDTOKN contains common procedures for generating a soap fault. Code will be added to display the fault message in CICS.

5.3. **CPPDTOKN**

CPPDTOKN contains common procedures for handling tokens. Two new sections will be added to allow programs to obtain and rewrite data stored with the token.

5.4. **CPWSFKEY (new)**

CPWSFKEY will be the working storage area used by CPPDFKEY.

5.5. **CPWSTOKN**

CPWSTOKN contains fields used for handling token. It is associated with procedure copylib member CPPDTOKN. A new field, SESS-STATUS, was added which will be used to store a status code along with the token.
5.6. CPWSWEBS

CPWSWEBS is the web service work area. Three new fields will be added to support the new hire application:

- CPWSWEBS-CONSISTENCY EDIT-SW
- CPWSWEBS-CONSISTENCY EDIT-STATUS
- CPWSWEBS-USER-TOKEN

5.7. CPWWEDB (new)

CPWWEDB will define the external which will be used as the communication area between web service PS006 and the "screen handler" PPWWEDB. It will replace the functionality of the BMS map for the WEDB function. It will contain an array of data elements and values.

6. DDL Members

6.1. PPPSUS Table (new)

PPPSUS is a new table which will hold suspended update data for the web EDB update. The following DDL members will be released for this PPPSUS:

- TSSUS00C – tablespace definition
- TBSUS00C – table definition
- IXSUS01C – index definition
- IXSUS02C – additional index definition
- PPPVZSUS – full table view definition

PPPSUS will contain the following column definition:

| SUS_BLOB | BLOB(1M) |

To contain the “blob” data, an auxiliary table must also be created. The auxiliary table for PPPSUS is PPPSUSX.

6.2. PPPSUSX Table (new)

PPPSUSX will be the auxiliary table containing the “blob” data for table PPPSUS. The following DDL members will be released for this PPPSUS:

- TSSUSX0C – tablespace definition
- TBSUSX0C – table definition
- IXSUSX0C – index definition

6.3. PPPTLA Table (new)

PPPTLA will be a new table containing template data stored for web EDB update users. The following DDL members will be released for this PPPTLA:

- TSTLA00C – tablespace definition
- TBTLA00C – table definition
- IXTLA01C – index definition
- PPPVZTTLA – full table view definition
6.4. **Indexes to Improve Performance**

The following database indexes will be added to improve performance of new queries.

- IXDIR06C creates new index UC0XDIR6 on the UC0DIR table
- IXIDB05C creates new index UC0XIDB5 on the UC0IDB table
- IXPNA05C creates new index UC0XPNA5 on the UC0PNA table
- IXPNX06C creates new index UC0XPNX6 on the UC0PNX table
- IXPNX07C creates new index UC0XPNX7 on the UC0PNX table

7. **Include Members**

7.1. **PPPVZSUS (new)**

PPPVZSUS will contain the layout for the new view PPPVZSUS_SUS.

7.2. **PPPVZTLA (new)**

PPPVZTLA will contain the layout for the new view PPPVZTLA_TLA.

8. **Bind Members**

The following is a list of the new package bind members which will be created for the new programs in the release.

PPCANHIR
PPPRGPEN
PPWWEDB
PS006
PS007
PS008
PS012
PS013
PS015
PS018
PS021
PS024
PS025
PS026
PS027

9. **Web Service Definitions (wsdl)**

See the section labeled “CICS Web Services” for a list of the wsdls for each new web service.
10. Control Table Updates

10.1. PPPCTT

To provide translations for valid code values returned by web service PS013, new rows in table PPPCTT have been provided in the form of transactions for program PPP004.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A38</td>
<td>EDB0118 A</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 B</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 C</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 D</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 E</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 F</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 G</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 H</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 I</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 J</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 K</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 L</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 M</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 N</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 O</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 P</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0118 R</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0130 M</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0130 S</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0244 N</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0244 Y</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0245 N</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0245 Y</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0246 N</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0246 Y</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0247 N</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0247 Y</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0248 N</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0248 Y</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0291 A</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0291 A</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0291 F</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0291 F</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0344 M</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0344 N</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0345 E</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0345 N</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0350 N</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0350 V</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0351 N</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0351 S</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0352 H</td>
</tr>
<tr>
<td>A38</td>
<td>EDB0352 N</td>
</tr>
<tr>
<td>A38</td>
<td>EDB1170 15</td>
</tr>
<tr>
<td>A38</td>
<td>EDB1170 17</td>
</tr>
<tr>
<td>A38</td>
<td>EDB1170 18</td>
</tr>
<tr>
<td>A38</td>
<td>EDB1170 19</td>
</tr>
<tr>
<td>A38</td>
<td>EDB1170 20</td>
</tr>
<tr>
<td>A38</td>
<td>EDB1171 15</td>
</tr>
<tr>
<td>A38</td>
<td>EDB1172 17</td>
</tr>
<tr>
<td>A38</td>
<td>EDB1173 18</td>
</tr>
<tr>
<td>A38</td>
<td>EDB1174 19</td>
</tr>
<tr>
<td>A38</td>
<td>EDB1175</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2008</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2010</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2010</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2010</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2010</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2011</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2011</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2011</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2011</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
<tr>
<td>A38</td>
<td>EDB2059</td>
</tr>
</tbody>
</table>
10.2. UC0BUN

The WHIR bundle will be added to the UC0BUN table with a single function, WEDB. Web service PS006 will set the function code to WHIR for a new hire process.

| BUN_SUBSYSTEM_ID | BUN_BUNDLE_NAME | BUN_BUNDLE_SEQ | BUN_FUNCTION_ID | B]
|------------------|-----------------|----------------|----------------|---
| EU               | WHIR            | 1              | WEDB           | P |

10.3. UC0CFN

Two rows will be added to the UC0CFN table. The WEDB function will be used for all EDB update web calls. WHIR will be used for new hires only and is defined as a bundle.

<table>
<thead>
<tr>
<th>CFN_SUBSYSTEM_ID</th>
<th>CFN_FUNCTION_ID</th>
<th>CFN_CICS_TRAN_ID</th>
<th>CFN_DTL_SCRN_PGM</th>
</tr>
</thead>
<tbody>
<tr>
<td>EU</td>
<td>WEDB</td>
<td>T001</td>
<td>PPWWEDB</td>
</tr>
<tr>
<td>EU</td>
<td>WHIR</td>
<td>T001</td>
<td></td>
</tr>
</tbody>
</table>

VI. Web Design

1. Technology Stack

EDBUpdate is a standard Java Model/View/Controller (MVC) web application. It uses Struts 2 for application control, JSP templates laid out using Tiles and the Struts 2 taglibs for its views, and an Employee and various other domain objects to model the business.

It accesses all business operations through a service-layer delegate/façade pattern. Each controller action that requires business operations calls upon a business delegate that has an interface type instance-level reference to the façade it needs to accomplish the business operation.
2. **Security and Authentication**

The web new hire will rely on PPSWeb to authenticate the user and generate a token number to be used throughout the user’s session. The token number is passed to the application using a secure cookie. The user id is passed as a parameter value.

The application ensures that the user has authority to access new hire system by checking against the RACF profile “WHIRE”. The user is sent back to PPSWeb main menu if the access is refused. The application replies on Struts 2 framework against any cross-site scripting.

3. **Screens**

The application is divided into 8 panels for collecting details about new hire:

- Employee Identification
- Personal Information
- Addresses and Disclosures
- Citizenship and Taxes
- Employment Information
- Appointments and Distributions
- Education
- Pay Disposition.

In addition to the above panels, the following panels provide added feature to the application:

- Summary
- Review Messages and Update
- Post-Authorization Notice
- IDOC

The user can navigate from one screen to another by using “Prev” and “Next” buttons at the bottom of each panel. The user can also jump to any specific screen by clicking on the link on the left side navigation bar.

3.1. **Employee Identification**

This is the starting point for a new hire process.

The following fields are available to the user for entry:

- First Name
- Middle Name
- Last Name
- Name Suffix
- Date of Birth
- Social Security Number

An employee Id. is assigned at the successful completion of this screen.

3.2. **Personal Information**
Information in this section is intended to identify the employee’s ethnicity, gender, and veteran status. The purpose of the requested information is to meet the University’s legal obligations as a federal contractor.

The following fields are available to the user for entry:

- Sex
- Ethnicity
- Disabled Status
- War Campaign Expedition Flag
- Armed Forces Medal
- Vietnam Flag
- Recently Separated Veteran Date
- Veteran Disability Status

### 3.3. Addresses and Disclosures

The screen captures the employee’s contact information including permanent address, campus address, telephone number and disclosure preferences.

The following fields are available to the user for entry:

- Address Type
- Home Address Line One
- Home Address Line Two
- Home Address City
- Home Address State
- Home Address Zip
- Home Address Province
- Foreign Postal Code
- Home Address Country
- Home Address Campus Release
- Home Address Organization Release
- Home Phone
- Home Phone Campus Release
- Home Phone Organization Release
- Spouse Name
- Spouse Name Campus Release
- Campus Address Line One
- Campus Address Line Two
- Campus Address City
- Campus Address State
- Campus Address Zip
- Campus Address Building
- Campus Address Room
- Campus Phone One
- Campus Phone Two

### 3.4. Citizenship and Taxes

This screen captures the employee’s citizenship status and federal and state tax withholding.
The following fields are available to the user for entry:

- Citizenship Status
- Country Of Residence
- Visa Type
- Work Permit End Date
- Income Code
- US Date Of Entry
- UC W-BEN Signature Date
- Federal Tax Marital Status
- Federal Tax Personal Allowances
- Federal Tax Additional Withholdings
- State Tax Marital Status
- State Tax Personal Allowances
- State Tax Itemized Deductions
- State Tax Additional Withholdings

### 3.5. Employment Information

This screen captures the employee’s employment information.

The following fields are available to the user for entry:

- Date of Hire
- Oath Date
- I9 Date
- BELI
- BELI Date
- Employee Relations Code
- Academic Programatic Unit Code
- Home Department Code
- Alternate Department Code
- Special Training Required Code
- Student Status
- Number Of Units
- Probationary Period End Date
- Next Salary Review Date
- Next Salary Review Type
- Intercampus Transfer

### 3.6. Appointments and Distributions

This screen captures appointments and distribution related to the employee’s employment.

Appointments are established to define the terms and conditions of employment with the University. Distributions are established to define how the employee is paid.

The following fields are available to the user for entry for an appointment:

- Appointment Number
- Appointment Type
- Basis
- Paid Over
3.7. Education

This screen captures the new employee’s education related information.

The following fields are available to the user for entry:

- Education Level
- Year Education Completed
- Highest Degree Institution Code
- Highest Degree Specialty Code
- Current Specialty Code 1
- Current Specialty Code 2
- Current Specialty Code 3
- Prior University Service Institution Code
- Non UC Prior Service Code

In addition to education, employee’s licenses and certificates information can also be entered.

The following fields are available to the user for entry related to licenses and certificates:

- Code
- Number
- Renewal Date
3.8. Pay Disposition

This screen captures the information regarding employee’s pay disposition.

The following fields are available to the user for entry:

- Check Disposition
- Online Earnings Statement
- Check Address Line One
- Check Address Line Two
- Check Address City
- Check Address State
- Check Address Zip
- Direct Deposit Bank Table
- Bank Account Number

3.9. Summary

This screen provides user with a single screen view of all of the information entered for the employee. From this screen, user may proceed to view consistency edit message or go back to any of the data entry screens.

3.10. Review Messages and Update

This screen presents user with the result of consistency edits process.

The messages are sorted by severity level. All 7-level severity messages are considered to be fatal and must be corrected. The user has the option to fix the error on the same screen, by clicking “Fix” button next to the message. User may also choose to go to the appropriate entry screen to correct the data entry errors.

The following describes the message level and severity:

- Level 7 – Fata error. Must be corrected
- Level 4 – Values changed due to system derivation
- Level 3 – Warnings
- Level 2 – Default value assigned

If there are no fatal consistency edits found, the user can proceed to post-authorization screen.

3.11. Post-Authorization Notice

PAN is a post-authorization system. It is used to send notifications to reviewers of PPS transactions. The user must enter comments explaining the new hire transaction.

The following fields can be entered on PAN screen:

- Priority – Normal/Urgent
- Comments

The user can add additional reviewers using one of the following ways:

- User id
- Email Address
- Locate by Last Name
From this screen, user can proceed to PPS Update. This will update the EDB database with the new employee information and generate IDOC form.

3.12. IDOC

This screen displays the available IDOC forms for the employee, separated by tabs on top.

The user can print the IDOC form, one at a time, by clicking on “Print IDOC” link.

The “Next” button ends the current transaction. A confirmation screen is displayed, with the link to Web EDB Inquiry system.

4. Model

4.1. Employee Object

The Employee object is the main business model object. It has instance level fields for each form input element on every view. It also has data structures to model the employee’s appointments, distributions, and licenses and certificates.

All fields modeled on the employee object that end up being posted to the back end are modeled as data elements, matching the data design of PPS itself.

4.2. Data Elements

The `edu.ucop.edb.domain` package defines the base Data Element type the application uses to communicate with outside services. It captures the basic fields of a data element:

- Number
- Value
- Occurrence Key
- Related Data

by defining accessor methods for each of these fields.

The EDB Update application extends this interface to provide methods the application needs to work with various PPS requirements. The extensions include:

- Setter on data element number
- Setters on value and related data
  Means for application to set fields to be passed to outside consumers
- Access to display metadata
- Access to a containing data element
  Needed for appointment and distribution-level data elements.
- Clone

The application must submit data elements to PPS for range value edits, consistency edits, PAN, and final commitment to the EDB. PPS requires different behavior for different data elements depending on which service the application is using. The data element modeled in EDB Update behave in a polymorphic manner to free the service layer from any special handling required to deal with different types of data elements.
The EDBUpdate data element also makes a distinction between getting its value when posting to PPS for range/value edits and when posting for consistency edit/submit to EDB. Data elements that post their values as an occurrence key will return their currently set value when the `getRangeValueEditValue` but null when their `getValue` method is called. Action code (DEN 0001) is an example of this type of data element.

To enable the polymorphic behavior, the application uses seven subclasses of the base EDB Update data element implementation class to fulfill PPS requirements for certain data elements. These include:

- **Appending occurrence key**
  This type of data element must append its value to its occurrence key when submitting to consistency edit, PAN, and EDB.

- **Related data special handling**
  This type of data element must post as its related data the value from another related data element. `Appointment.rateAmout` is an example of this type. It uses the value set on `Appointment.rateType` as its related data.

- **Parent element occurrence key**
  Elements of this type submit their occurrence keys as the value of a parent data element for consistency edit/submit only. For example, license code (0711) is the parent of license number (0712) and license date (0718). 0712 and 0718 must set their occurrence key to the value of their related parent data element.

- **Truncating data element**
  These elements store a long value for display but must truncate to a shorter length when posting to range/value and consistency edits. All code-based fields are of this type, for instance, title code posts its four-character code but stores the concatenation of the title code and its title for display. The truncation length is an instance level field passed on construction.

- **Occurrence key/Value-truncating**
  A combination of both of the above: the value must be truncated, and posted as the occurrence key. The license code (0711) uses this type of data element model.

- **Value stripping**
  These data elements must remove display formatting when posting to the consistency edit and submit services. `Employee home phone` (0210) is an example.

The following table summarizes some of this behavior when the getter for range/value edit, consistency edit value, occurrence key, and related data:

<table>
<thead>
<tr>
<th>get:</th>
<th>RVEValue</th>
<th>Consistency editValue</th>
<th>OK</th>
<th>Related Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDB Update DE</td>
<td>Value</td>
<td>Value</td>
<td>OK</td>
<td>RD</td>
</tr>
<tr>
<td>Appending OK</td>
<td>Value</td>
<td>Null</td>
<td>OK + value</td>
<td>RD</td>
</tr>
<tr>
<td>Related Data special handling</td>
<td>Value</td>
<td>Value</td>
<td>OK</td>
<td>Value from related DE</td>
</tr>
<tr>
<td>Multipart</td>
<td>Concatenation of component values*</td>
<td>Concatenation of component values1</td>
<td>OK</td>
<td>RD</td>
</tr>
</tbody>
</table>

---

1 Each component is right padded to its defined length with spaces.
<table>
<thead>
<tr>
<th>Parent Element occurrence key</th>
<th>Value</th>
<th>Value</th>
<th>Value from parent element</th>
<th>RD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truncating DE</td>
<td>Truncation of current value</td>
<td>Truncation of current value</td>
<td>OK</td>
<td>RD</td>
</tr>
<tr>
<td>Occurrence key/value-truncating</td>
<td>Truncation of the current value</td>
<td>Null</td>
<td>Truncation of the current value</td>
<td>RD</td>
</tr>
<tr>
<td>Value-stripping</td>
<td>Value</td>
<td>Value stripped of formatting characters (chars='(', ')', ',', ')')</td>
<td>OK</td>
<td>RD</td>
</tr>
</tbody>
</table>

**KEY**
- RVE = Value = range/value edit value
- OK = Occurrence key
- RD = Related data

### 4.2.1. Employee Level

The majority of data elements used by EDB Update are member fields of the EDB Update employee object. Internally each data element is stored in a map of data element number (as string) to the data element.

When a new employee object is initialized, the metadata manager creates a new instance of this value map.

### 4.2.2. Appointment/Distribution Level

Appointments are modeled as separate data stores within an employee object. Each employee instance contains an appointment set object, which manages the appointments, including tracking available appointment numbers, enforcing business rules, and data element list retrieval for range value, consistency edit, and submit operations.

Each appointment similarly contains a distribution set that provides the same functionality as the appointment set does to the employee object, including managing

### 4.2.3. Data Element Display Info

Each data element in EDB update has an associated display info object that captures metadata about how the data element is presented, including:

- Its display type
  - Enumerated by `edu.ucop.edbupdate.domain.DisplayType`
    - RADIO
    - DROPDOWN
    - AUTOFILL
    - TEXTFIELD
    - MULTIPART
    - DISPLAYONLY
- Name of the view it is used on
- Input name
- Name of list used for dropdown, radio, or autofill population
- Full name, used as label in presentation
An optional hint, which can include a mask and a css style class

When the application builds the forms used by the consistency edit view for each message, the application relies on this metadata to build the correct type of input for each data element related to a particular consistency edit message on the fly.

4.2.4. Metadata Manager

EDB Update depends on a good deal of metadata to work properly. This information is captured in the edu.ucop.edbupdate.newhire.util.DataElementMetadataManager class. It includes:

- A string constant for every data element number used by a field on the Employee object.
- A string array constant containing all the data elements found on each data entry view in the application.
- A set of string array constants for each set of pivot fields (see below for discussion of pivot fields).
- Methods to look up both of the above by current action.
- A method to initialize the data element map needed by the employee object to store data elements.
- A method to add any campus-customized additional data elements to the base employee data element map.
- A method to lookup any employee-level data element information by data element number.
- A method to get the set of data elements for any appointment or distribution, identified by its number.

5. View

The application uses JSP templating, struts tags, and extensive use of Strut’s OGNL ability to conditional fill dynamic values in the templates. There are no Java scriplets or page imports on any template.

5.1. Tiles

The application uses Tiles to compose the templates from reusable subcomponents, all defined in tiles.xml. The base New Hire layout includes

- headContent
  All information contained with the <head> element of the DOM, including all script links, script fragments, and a meta tag for content type.
- bodyHeader
  Top band visible to user, containing the application name, links to the menu main and log off, and a user-specific welcome message
- functionBar
  The module name (e.g. New Hire), and suspend/restore transaction
- messagePanel
  Usually empty, contains application-level messages to the user, such as when they try to navigate to a view not allowed by their current transaction state.
- content
  The main data entry portion of each view.
- errors
  Usually hidden, embedded within the content tile, contains range/value edit error messages.
- navBar
  Displays navigation links to all views in the application.
- footer
  Displays UCOP’s address, the copyright notice, and the current instance’s version.
5.2. Tiles Schematic

5.3. Controller

EDB Update uses the Struts 2 framework as its control layer. All action classes subclass NewHireBaseAction, which contains behavior and attributes shared by all the actions. Each view on the application has its own action.

5.4. Struts Configuration

5.4.1. Namespacing

The New Hire module of EDB Update is configured in its own Struts namespace, “newhire”, and all its action mappings are contained in the “newhire.xml” struts config file. All New Hire action urls are prefixed with the string “newhire”, so for instance the URL for the Education view is /([Context root]/newhire/education.

5.4.2. Method Invocation

EDB Update enables Struts’s struts.enable.DynamicMethodInvocation option, which allows non-mapped, public-visible, String-returning methods to be invoked as action methods, which lessens the number of entries in the configuration file.
5.5. Struts Themes

The application uses the struts “simple” theme to suppress struts’ auto-wrapping of all struts form tag elements in a two-celled table row. The application uses a custom theme, radiovertical, which formats radio buttons vertically. The template is defined in the /templates/radiovertical/radiomap.ftl file. The application uses it on the following views: addresses and disclosures, citizenship and taxes, pay disposition, and personal information.

6. Pivot Fields

Some fields toggle the presentation of other fields. For example, on the Personal Information view, when the user select “Yes” for the “Is the Employee a US Veteran?”, five more data elements that capture veteran status information become visible. When the user selects “No”, these fields disappear, if presently visible.

For EDB data element fields, their data element number is used as their identifier, for non-EDB pivot fields, the app uses a synthetic unique identifier internally. These values are defined as constants in edu.ucop.edbupdate.newhire.util.DataElementMetadataManager. Currently these values include:

```java
public static final String VETERAN_STATUS = "pivot.vetStatus";
public static final String US_CITIZEN = "pivot.usCitizen";
```

Each view that has one or more pivot fields has a statically-initialized array of edu.ucop.edbupdate.newhire.util.PivotFieldInfo that capture all pivot field information the application need to process these fields. The information includes the pivot field’s ID, its trigger value, and the list of additional data elements that will be present when the pivot field is set to its trigger value.

NewhireBaseAction defines an abstract getter method on this data:

```java
protected abstract PivotFieldInfo[] getPivotFieldInfo();
```

Subclass actions whose views don’t contain pivot fields return null for their implementation of this method.

The Employee class defines a getter and setter for each additional field. Each view’s template defines input fields for each additional field. Javascript logic handles the conditional visibility of these fields on the browser. Each field references the named field on the employee object, so the Struts framework automatically updates the additional field values if submitted to the action.

On update, since the additional fields are not included in the view’s set of normal data element numbers (as returned by DataElementMetadataManager.getScreenDENs( getCurrentAction() ));—see Range Value edit) they must be added by first getting the current action’s set of pivot field info objects, and if that array is not null, by checking the current value for the trigger field against each trigger value in the array of pivot field info objects. This check uses the

```java
Employee.getPivotFieldValue( String pivotFieldId )
```

method. This method is backed by a mapping from pivot field ID to the current value for that field:

```java
private Map<String, Object> pivotFields;
```

This additional data structure is populated at the setter for each pivot field on the Employee object. If the current value matches a trigger value, then the additional data element numbers are copied into the array of data element numbers returned by the getScreenDENs call.

Since the additional fields are defined by the Employee class as regular data element fields, they are automatically included for consistency edit and submit operations.
When the user edits a consistency edit error on the consistency edit view however, the application must determine if the user edited a trigger value in such a way that the additional fields are no longer relevant and should be reset to null. The Employee object exposes a public method, checkConsistencyEditPivotStatus() which performs this task. For each data element involved in a consistency edit update, it checks to see if it is a pivot field, and if it is, if its current value matches the trigger value. If it does match the trigger value, it sets all related data elements to null.

### 6.1. Pivot Field Inventory

<table>
<thead>
<tr>
<th>View</th>
<th>Pivot Field</th>
<th>Form Name</th>
<th>EDB DE</th>
<th>Trigger Value</th>
<th>Add’l DEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal Information²</td>
<td>US Veteran</td>
<td>emp.vetStatus</td>
<td>No</td>
<td>“Yes”/’Y’</td>
<td>0344, 0345, 0350, 0381</td>
</tr>
<tr>
<td>Address &amp; Disclosure</td>
<td>Permanent Address</td>
<td>emp.addressType</td>
<td>Yes (0291)</td>
<td>“Foreign Address”/’F’</td>
<td>1118, 1119, 1120</td>
</tr>
<tr>
<td>Citizenship &amp; Taxes</td>
<td>US Citizen</td>
<td>emp.usCitizen</td>
<td>No</td>
<td>“US Address”/’A’ “No”/’N’</td>
<td>0110, 0184, 0707</td>
</tr>
<tr>
<td>Appointment</td>
<td>Appointment Type</td>
<td>myAppointment.type</td>
<td>Yes(0291)</td>
<td>“5 – Academic”/”5” and “7 – Part Year Career”/”7”</td>
<td>Appt 10: 2010, 2011</td>
</tr>
</tbody>
</table>

### 7. Range Value Edits

The action of each form on the eight main data entry views is “[current action name]!update”, which refers to a method in the base action class. This method uses the get current action method implemented by all actions of this type, passing the current value to the metadata manager, which keeps a static array of all data elements found on each view. Using this array, the base action class update method asks the employee object for a list of range value data elements, which are then sent to the range/value service.

If the any element fails RVE, its data element is added to a list that is written into a special error div on each page. There is an on document ready script that detects content in this div (displayAndHighlightRangeErrorsValueIfAny() in common.js), that uses each data element number to find the inputs by title attribute (every input element’s title attribute is set to the field’s data element number) failing RVE for error markup.

### 8. Navbar Invocation

Each navbar’s element’s on click is attached to a javascript function (enableNavBarUpdating() in common.js) that submits the current page’s form, adding a hidden input field that contains the desired destination of the user, keyed by the same name.

---
² The initial value is set to ‘N’ by javascript. See personal-information.js.
³ 0109, citizenship type is hidden when US Citizen is “Yes”, but the value is submitted in either case, and is set to ‘C’ for US citizen when the trigger value is “Yes”.
⁴ 2011 is only displayed when the appointment type is “5”. 2010 is display for both values.
as the next and previous navigation inputs “nav”. Each of the possible forwards are named as global forwards in the struts mapping file.

9. **Next/Prev Invocation**

The “Next” and “Prev” buttons at the bottom of each data entry view are submit type inputs, with a name of “nav” and a value that matches its label.

10. **Consistency edits**

EDB Update groups consistency edit messages by severity level, then by consistency edit message number. The consistency edit façade returns a bare list of consistency edit items with one entry for each data element that is associated with a consistency edit.

The application allows the user to fix any consistency error on the review screen. The application accomplishes this by building a form for each message that includes all related data elements on the fly. Each data element’s display info provides the information needed to generate the proper input element for it.

Employee-level data elements have form inputs named to match their employee object field, so that Struts framework will automatically set their values on submit. The consistency edit action class must set appointment or distribution level fields itself. Each appointment/distribution data element shares a form input name, “variableDETypeValue” which Struts sets as an array of strings on the action. The label cell for each of these fields contains a hidden input with the shared name “variableDETypeDen” with its value set to the current data element number for the field in question. In this way, when the action is processing the update, there are two arrays, with the data element number and its corresponding value at the same index in each. The update logic tests for non-null, non-zero length state of the element number array as the signal that there are some appointment/distribution fields in the current update. This logic must also handle the single special case of the FAU field on an distribution, since it is a composed field. The template logic tests each data element info for a non-null value in the myOwner field, which for appointment and distribution fields, holds a reference to the appointment or distribution object cached in session memory (via the employee object’s AppointmentSet and each appointment’s DistributionSet) as the signal that the variable number/value arrays should be written into the current form element.

Each consistency edit item listed on the review screen has a fix button which invokes an AJAX call to the application with the message number. The application responds with the form for that message as a DOM fragment.

The application keeps track of the current set of outstanding consistency edits so that with each new invocation, it can report to the user the number of fixed and added messages. The consistency edit action caches the current set of consistency edit message keys in the user’s HTTP session to enable this tracking.

11. **PAN**

The post authorization notification view allows users to add additional reviewers and see the formatted PAN notice as it will be sent.

11.1. **Add recipient**

A user can add an additional review by User ID, by email address, and by lookup on the user’s name. In all three cases the application calls checkForExistingRvwr, which scans the table of current reviews to make sure the reviewer the user is trying to add isn’t already in the table.
11.1.1. By User ID

The application makes an AJAX call to the action’s lookupByUID method, which invokes that service on the façade. The action method enforces some field-level validation: the entered user ID must not be empty or greater than eight characters long. If the user is found, the template fragment returned by the action is the entire contents of the reviewer display table, rebuilt with the new reviewer appended to its end. This makes it easier to alternate the shading of each table row.

11.1.2. By Email Address

When the user enters an email address, a field-level validation check determines if the address contains an “@” sign. If it does, and this address isn’t already in the table (via the checkForExistingRvwr) call, the application calls the postSelectedRvwr function, which invokes the action’s addReviewer method, which also returns the rebuilt table in the same manner as adding a reviewer by user ID.

11.1.3. By Name

When the user enters a name into the Last Name input element and clicks the “Find” button, the application invokes the action’s findNameInDirectory method, which uses the service layer’s PAN operation that finds that name in the list of all reviewers. It will return the closest match as the first item in a list of reviewers ordered by last name. The application displays this list in a jQuery grid. The user can pick the number of items to display using the control on the grid. The choices are ten, twenty, and thirty.

The grid supports standard paging controls: next, previous, first, and last. The grid takes the number of pages and the current page as options in its constructor. Since the service doesn’t support returning how many pages of data are available, but does indicate when the current results are at the beginning or end of the dataset, the application always sets the number of pages as three, and if at the beginning, sets the current page at one, or if at the end, then sets the current page as three, or in all other cases, sets the current page as two. This causes the grid to always activate the appropriate set of paging controls (e.g. when on page one, the first and previous buttons are disabled.)

When the user selects a row in the grid, and then clicks the “Select” button, the application runs the existing reviewer check then, if unique, invokes the post reviewer function, passing all three displays fields (user ID, name, and email address.)

11.2. View PAN Notice

The PAN view includes a link entitled “PAN Notice”. This link points at the action’s htmlDetails method. The post_auth_notification.js document ready function binds the anchor element to a click handler that selects the anchor’s href attribute to get the URL to the action method, and opens a new popup window for the results. Each window gets a unique ID based on the timestamp at the time the user clicks on the link, so that the user can open more than one at a time. The service returns the PAN notice as marked up HTML with the original PAN service invocation. The action stores the data transfer object it gets from the service layer in the user’s session so that it can be retrieved should the user invoke this feature.

12. IDOC

The IDOC view allows user to view and print IDOC forms generated for the new employee.

The Jasper Reporting tool has been used to create pdf version of IDOC form for printing.

The web service sends the generated IDOC forms as pre-formatted for a mainframe line printer. The following formatting characters are taken care of by the IDOC print function:

“1” – Indicates start of new page
“0” – Indicates skipping a line (Blank line)
13. Templates

Templates are sets of arbitrary data for related to one of three views that users can save for reuse. The views that support the template feature include:

- Addresses and disclosures (campus address fields)
- Appointments (all fields)
- Distributions (all fields)

The application models each template as a `edu.ucop.edb.domain.Template` object containing:

- A string name
- An enumerated template type:
  - CAMPUS_ADDR
  - APPOINTMENT
  - DISTRIBUTION
- A list of `DataElement` objects storing the templates actual data

Browser interaction with the template action is through AJAX calls. All template-related javascript is contained in `template.js`.

13.1. Using a Template

When a user loads a saved template, the application sends the data as a JSON array of data element number/value pair objects for all three types of templates.

If the template is a campus address type, the javascript function that populates the template fields iterates across all data elements returned in the JSON object, using a jQuery each() function. Since every input element stores its related data element number in its title attribute, the JS can find the input element for each item in the JSON object by selection on that title, which returns a div that setting the element’s value. Since the template data sent by the application does not capture the input type of each element, the logic attempts to find each item first as a text field, then as a dropdown, and finally as a radio.

If the template is an appointment or distribution type, the javascript logic must rely on a different mechanism, since the data element numbers of each appointment or distribution will vary by appointment/distribution number for the same field. Therefore, instead of a simple array of DENs, the javascript must use a mapping of the DENs for appointment 10/distribution 11 to unique string IDs for each input element. The template data is stored using the base DENs. In the `each()` implementation, the code looks up the element’s ID in the map by the base DEN, then uses that ID to select the correct element. As with the campus address template, the logic attempts to first set the value as a text field, then a dropdown, and finally as a radio.

There is one special handling case associated with setting the value for the appointment title code (DEN 2306). Since it’s an autofill text input, and only the four-digit code is actually stored with a template, the code must iterate across the field’s autofill value array to find the full translation for a given saved title code value.

Distribution templates require two special handling cases. The first is for the FAU field, since it is displayed as a multi-part component. The value is a series of integer indexes and the saved value. The distribution use template function first finds all input elements that make up the components of the FAU, selecting on ID beginning with a known string (“fullAccountingUnit”). The logic then accesses each element of the selection array by the stored index passed back by the template action to set its value.
The second special handling case is parallel to the appointment title code case: the distribution DOS code is stored as a three-character value, but displayed as the code plus its translation. The code iterates over the DOS code autofill value array to find the current code’s translation.

### 13.2. Saving a Template

Template values are posted to the application as a querystring. The querystring is build up by the saveTemplate function. If the template is a campus address type, the code iterates on a constant array of all the data element numbers (DENs) that can be included in a template, using a jquery each() call, selecting each input element by the input element’s container div, which has the data element number of the input element set as its title attribute.

Appointment templates need to use the inverse data structure from the one used when restoring a template, which is a map from element ID to base appointment data element number, to find each input element. The querystring is build using the base DEN as the key for each element.

As in the restore case, the title code element requires special handling. Only the first four characters of the input value is saved.

Distribution templates parallel appointments in the use of the element ID to distribution 11 DENs. The special handling of the FAU inputs mirrors the handling when restoring: the code selects all component inputs by ID substring match, then builds the querystring value string by concatenating the current index of the iteration function with the value found at that index in the array of selected elements. The special handling of the DOS code field also parallels the appointment title code case: the code truncates the current value to the first three characters of the current value.

### 14. Application Properties

All application properties are loaded by the `edu.ucop.edbupdate.newhire.utilApplicationPropertiesLoader` class, which implements the `ApplicationPropertyStore` interface. This class is defined in the Spring application context as a singleton and injected into the base action class, which then makes it visible to all subclass action templates via a public getter.

#### 14.1. Code Translations

The application property loader has an instance field of type `ITranslationService` which provides its translation service. The application has a web service-enabled implementation and a hardcoded implementation for testing and development without necessitating use of external services. The application property loader builds key/value lists for each data element that has a code translation list. The templates tags access these lists via the application property store instance visible on the base action class.

#### 14.2. Externalized Properties File

The application property loader also handles reading in the externalized application properties, which it stores in the `EDBUpdateProperties` object. These properties include:

- `webppsserver`=[url of PPS Web Menu app to link to]
- `webppscontextroot`=[Context root of PPS Web Menu app]
- `session_timeout`=[user session duration, in minutes; overrides value set on app server]
- `securityprefix`=PPSW
- `reload.app.data.timeofday`=[time that the app property loader should attempt to reload its data]
The location of this property file must be passed to the JVM as a command line custom property keyed by “EDBUPDATE_PROPERTY_HOME”.

### 14.3. Consistency Edit Severity Levels

Each instance of PPS can have different names for each consistency edit severity levels. These names are stored in the CTT as regular code translations. The application property loader builds an order list and a lookup map of each severity level and its translation. These data structures are used by the consistency edit action class and its template.

### 14.4. Campus Customizations

The application checks the the same directory indicated by the JVM system property “EDBUPDATEPROPERTY_HOME”, where it looks for a file named CampusCustomizations.properties. The application expects each line in the file to contain a known key value, a colon, and a customization value. Any line that begins with a hashmark (“#”) is treated as a comment and ignored.

All customizations are stored in an instance of edu.ucop.edbupdate.newhire.util.CampusCustomizationData, which contains accessors for all customizable properties. The application property singleton object contains an accessor to this object, which allows actions and templates to read any of its properties when needed.

The supported keys include:

<table>
<thead>
<tr>
<th>Key Name</th>
<th>Multiple Occurrences Allowed?</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>codeFilter</td>
<td>Yes</td>
<td>Drop down code filtering</td>
</tr>
<tr>
<td>allowPanEmail</td>
<td>No</td>
<td>Enable add PAN reviewer by email</td>
</tr>
<tr>
<td>nonCitizenDENs</td>
<td>No</td>
<td>Define visible foreign visa data elements</td>
</tr>
<tr>
<td>displayRegisteredUnits</td>
<td>No</td>
<td>Enable display of the registered units data element (0237)</td>
</tr>
<tr>
<td>displayPayDisposition</td>
<td>No</td>
<td>Enable display of the pay disposition view</td>
</tr>
<tr>
<td>payDispositionDENs</td>
<td>No</td>
<td>Define visible pay disposition data elements</td>
</tr>
<tr>
<td>displayHomeDepartmentCode</td>
<td>No</td>
<td>Enable display of the HD code data element</td>
</tr>
<tr>
<td>fau</td>
<td>No</td>
<td>Define the component parts of the full accounting unit data element</td>
</tr>
<tr>
<td>additionalField</td>
<td>Yes</td>
<td>Defines an arbitrary employee-level data element to add to the application</td>
</tr>
</tbody>
</table>

### 14.5. Drop Down Code Filtering

Any drop down element in the application can be filtered for the code to be displayed. For each item of this key, the application expects a four-digit data element number, which corresponds to the drop down to filter, the pipe character (“|”), and a comma-separated list of allowed values. When the application loads the translation service data, it checks each data element number to see if it is included in the set of data elements being filtered by this customization. If it is, it restricts the code values to those found in the customization.
14.6. Allow Add PAN Reviewer by Email

The application expects a value of “true” or “false” for this property. If false, the application suppresses the add reviewer by email elements on the PAN view. The template reads the value via the application property object and its campus customization data object, as described above.

14.7. Visible Foreign Visa Data Elements

The application expects a list of data elements numbers drawn from the complete list: 0109,0110,0143,0184,1168,1169,1170. The application will only display the listed elements when the user is working with a foreign visa employee. The application adds each listed data element in the customization file to a Set which it exposes as a public field. On the citizenship_and_taxes.jsp template, each input element is surrounded by a struts if tag, which tests to see if the current data element number is contained in this set, only displaying the data element markup if it is present in the configured set. The default is to display all data elements listed above.

14.8. Display Registered Units Data Element

The application expects a value of “true” or “false” for this property. If false, the application suppresses the registered units data element on the employment information view. It defaults to true.

14.9. Display Pay Disposition

The application expects a value of “true” or “false” for this property. If the property is false, the application suppresses the entire Pay Disposition view by removing it from the navigation bar’s set of links, and by pointing the education view’s “Next” at the summary, and the summary’s “Prev” at education, via conditional struts tags checking the current value set on the campus customization data.

14.10. PayDisposition Data Elements

If the pay disposition view is not suppressed by the above customization, the application will check for display of the following data elements, in the same manner as with the foreign visa data elements: 0201,0203,0230,0266,0301,1124. The default is to display all data elements in this list.

14.11. Display Home Department Code

The application expects a value of “true” or “false” for this property. If the property is false, the application suppresses the display of the home department code (0114) on the employment information view, using conditional struts logic on the employment_information.jsp template.

14.12. Full Accounting Unit Formatting

The application supports breaking the FAU into arbitrary-length sub-components. The application parses this item looking for:

<length>:<label>;<length>:<label>;

with the length specifying the size of the component, and the label the string to use as the component’s label. The components are displayed in the order they are specified in the customization string.

14.13. Additional Data Elements
The application supports additional arbitrary data elements. The customization requires at least the data element number and its label; the customization also supports truncation length, hint type, and input mask type. For each valid entry in the customization file with the `additionalField` key, the application adds the data element to the employee object’s value map, and adds them to the employment information view by iterating on the employee object’s additional data element list. All additional data elements have input type `textfield`.

15. **Reload daemon**

The application property loader refreshes its data on a daily basis, using a daemon thread. The items it reloads include:

- Campus customizations
- Code translations
- The consistency edit severity level name list
- The application parameter lookups
- The externalized property file

It will sleep until the following day at the time of day specified by the `reload.app.data.timeofday` value found in the external property file. If it fails to reload any type of data, it will retry in five minutes until it successfully completes its reload tasks. It synchronizes on a data store object for each step ensuring that any other thread will not read the data store as it refreshes.

16. **Javascript Minification**

16.1. **YUI Compressor**

There are 2 ways you may run YUI compressor.

1. The project contains a top-level, non-distributed directory, scripts. This directory contains bash shell scripts:

   - `css-minify.sh`
   - `js-minify.sh`

   For compressing the CSS and javascript files in the application. They should be run from within the `[projectRoot]/WebContent` directory.

2. Using the ANT script

   The project contains, at the top level, `build.xml`. This file has the required ANT script to run the YUI compressor. The target name to run is “compressAll”. This task will concatenate all the CSS and JavaScript file and create the compressed file.

   The property value, defined as `<property name="compressedFileVersion" value="1.2.008"/>` should match the current application version.

16.2. **Name Convention**

The minification of both the application javascript and stylesheets relies on the current application version as defined in the `Version` class. The script that minifies reads this value using the `main()` defined on the `Version` class, and the `headContent` template reads the same value as a static method call. Versioning these files ensures that when the user accesses the application after an upgrade, the newer versions of each file will load.
17. **Transaction Management**

17.1. **Session Suspend**

EDB Update supports caching of a user’s incomplete hire transaction under following conditions:

1. The user can invoke a suspension manually.
2. Auto-suspend by the system when the user’s HTTP session times out.
3. Auto-suspend by the system when it encounters unrecoverable run-time error.

In all cases, the suspend action creates a `SuspendSessionStruct` which holds the currently loaded employee, and the current action’s `NavigationOrder`. This object is serialized to a byte array, which is the input to the suspend façade.

17.2. **User Invoked**

A user can suspend a transaction at any time after IID assignment. The mechanism works the same as in the auto suspend case.

17.3. **Auto Suspend**

The head content tile template writes the current HTTP session timeout value (`pageContext.request.session.maxInactiveInterval`) as a javascript global variable. An on document ready function sets a timeout on the window with a callback to a function that invokes the suspend action via an AJAX call. The interval on the window timeout is the session max interval minus a short period defined in `common.js` by the javascript global `sessionWarnInterval`. This interval allows the application time to warn the user, and to suspend the transaction before the java session expires.

If the window timeout fires, the event handler displays a modal dialog warning the user that the session will soon time out. The function that opens the modal sets another window timeout that is the session warning interval minus three seconds. The three seconds give the application the time it needs to successfully complete the suspend before the application invalidates the user’s HTTP session. If the user chooses to refresh the session, the javascript invokes the refresh method of the suspend action, which is a no op that servers merely to reset the java session timer on the application side. If the user does not respond, and the current window timeout fires, the javascript invokes the autosuspend method on the suspend action.

Auto-suspend, also, is invoked when the application encounters any “uncaught java exception”. This could be the result of web services becoming unavailable or due to run-time errors.

17.4. **Restore**

The service requires both the employee ID and the IID timestamp to uniquely identify a suspended session. When the user clicks on the suspended transactions link, the application makes an AJAX call to the restore action’s list method. Each suspended transaction includes two hidden fields in the same cell as the Use and Delete buttons which have names of the form “session_employeeid_[ix]” and “session_timestamp_[ix]” where [ix] is a zeroth-based indexing of each suspended transaction. The Use button has the same index value set as its id attribute. On bind of the restore handling function, the binding function (bindIWantToRestoreButtons()) selects the employee ID and timestamp values from the hidden fields and includes them in the bind options to the handler. The restore handler then just passes the bind options as received as JSON data to the AJAX function, which ensures the application action class will have the two attributes corresponding to the suspended transaction the user desires to restore.
18. **PPS Web Main Menu**

The existing PPS Main Menu application will be modified to support the new application. A new link will be added to the main menu named "EDB Update System – New Hire”.

19. **Help**

The Web New Hire application provides users with field level help on all screens. Each field level help item is referenced by the EDB element number. When user requests help for an element (by pressing F10 key or by clicking on the field level), help text is displayed in a pop-up window. All the fields for which help is available is indicated on the screen with a broken red underline.

The application is provided with the base help files included in the distribution code. Application is also designed to override base help with customized help. An external EDBUpdate.properties file defines the location of customized help files. Any help file found in this directory will take precedence over the base help files.

Help files are named nnnn.txt, where nnnn is the data element number.

Help file (nnnn.txt) will have help text in “Textile mark-up language”. This language has some simple mark-up tags and is easy to learn and use.

In order to customize help for data element nnnn, create a file called nnnn.txt in the “help.directory”. Using any text editor (such as Notepad), create the required help text in the file.

Campus help is scanned automatically for special characters (embedded scripts etc) for security reasons. The application (textile utilities) picks it on a real time basis, and converts to html. The html is then rendered to the user.

The default base style (CSS) is used to display help.

20. **Field-Level Manipulations**

20.1. **Field-Level Business Rules**

EDB Updates uses a suite of javascript functions to enable field-level manipulations of data to enforce PPS business rules.

20.1.1. Uppercase Alpha Entry

All alpha input fields convert lowercase characters to uppercase by binding a JQuery plugin, Best Upper v1.0, to these fields, in the document ready call in common.js.

20.1.2. Appointment Rate Code/Rate Amount

The application uses custom JavaScript to enforce several business rules regarding appointment rate codes, appointment rate amounts, and associated distribution pay rate fields. The first bound function, `applyRateCodeFormatting`, in `appts-and-dists.js`, prevents entry into the rate amount field until the user picks a rate type. On change of the rate type, it clears the appointment rate amount and all related distributions. This function also binds an onblur event on the rate type field that invokes a padding function, `rpadRateAmount()`, in `common.js`, which right pads the appointment rate amount or distribution pay rate to the required length depending on the selected rate type. The function also binds the keydown event to `formatRate()`, which prevents users from typing anything but numbers and decimal points, and restricts the length of the whole number and decimal portion of the value depending on the currently selected rate type.

20.1.3. Percent Formatting

The document ready call in `common.js` also binds a percent formatter on the keydown event to all percentage fields in the application. This formatter works in a similar way to the rate amount formatter, preventing the user from typing anything but numbers or the decimal point. It also automatically fills in values in certain cases, such as when the user types a leading one,
where the logic fills in the decimal place and the trailing zeros, when the user types a leading zero, where the logic fills in the decimal point, and when the user types a decimal point, where the logic prepends the zero.

20.1.4. Autofill

Certain input fields on application views improve the user experience by suggesting values as the user enters characters. This feature employs the jQuery Autocomplete plugin, v1.3. It requires an array of values for each autofill field. These values come from the translation service. There is an autofill action that fills a template with a javascript array for each set of translations used in the application. This action is invoked by reference as a script tag in the headContent template, so that the browser loads it as javascript. Each array in the file becomes a global javascript variable. The application binds each field requiring this behavior to its corresponding array in a document ready call in common.js. The bind function explicitly finds each input element by either name or id, linking it to the named array contained in the autofill template.

21. Navigation Restrictions

There are certain views in the application that the user cannot access unless business requirements are met. For example, a user cannot navigate to the Post-Authorization Notification (PAN) screen until all level seven consistency edit messages are cleared.

The application enforces user navigation restrictions through a custom Struts interceptor, IllegaStateInterceptor, which is configured as the last interceptor after the Struts pre-built defaultStack. There are two enums defined in the application that track state the application needs in order to enforce these restrictions. They are:

- NavigationOrder
  Defines a constant for every view the user can visit.
- State
  Defines a constant for every state a transaction can be in

The two are linked by a Set<State> instance-level field on Navigation order, which contains all the legal states that allow navigation to each view.

The interceptor checks the current state against the desired view, indicated by the NavigationOrder returned by the current action’s getCurrentAction method. This abstract method is defined in NewhireBaseAction, so all of its subclasses respond to this message. If the interceptor sees that the destination is not allowed based on the current state, the interceptor short-circuits the response cycle, forward the user to the previous destination, which is captured in the user’s session by the constructor to NewhireBaseAction.

The following diagram illustrates relationships between the various new hire states:
22. **Textile Mark-up language**

Textile is a lightweight mark-up language. Textile converts its marked-up text input to valid, well-formed XHTML. The following are some of the common tags used. For a complete list of mark-up tags available, please refer to the product website [http://textism.com/tools/textile](http://textism.com/tools/textile).

Note: Tags must start in column one. Text file should not contain any non-printable characters.
22.1. Textile Tags

22.1.1. Headings

Use h tag with a number.

Examples:

h1. Your first-level heading
h2. Your second-level heading
h3. Your third-level heading

22.1.2. Bold

Use * around the text/word

Example:

*your text here*

22.1.3. Emphasized

Use _ (underscore) around the text/word

Example:

_your text here_

22.1.4. Bullets

Use *s based on level

Example:

* your text here for bullet
** your text here for sub-level bullet

Number

Use #s based on level

Example:

# your text here for numbering
## your text here for sub-level numbering

22.1.5. Web Link

Use “Click here”: followed by the link

Example:

"Click here":your_web_link_address