Table of Contents

CHANGE HISTORY .................................................................................................................................. 4
IMPLEMENTATION SUMMARY .................................................................................................................. 5
  Service Oriented Architecture User Interface .................................................................................. 6
  Web Services ..................................................................................................................................... 6
  User Account Manager ...................................................................................................................... 7
  Constituent Transaction Manager ..................................................................................................... 8
  Admissions API ................................................................................................................................. 11
POST INSTALLATION SETUPS ................................................................................................................ 13
  Verifying Web Service Security Settings ......................................................................................... 13
  Setting Up User Accounts ................................................................................................................. 15
  Performing a Security Analysis ........................................................................................................ 17
  Publishing Web Services .................................................................................................................. 18
  Refreshing the HCM Registry Cache ............................................................................................... 18
  Setting Up Integration Broker .......................................................................................................... 19
  Verifying Campus Solutions SOA Framework Setup Entries .......................................................... 20
  Setting Up Logging .......................................................................................................................... 22
  Verifying Constituent Transaction Manager Setup Entries ............................................................. 23
  Configuring AAWS ........................................................................................................................... 26
EXTENDING AAWS .............................................................................................................................. 35
USER INTERFACE CONSIDERATIONS ..................................................................................................... 49
  Discussion of the User Interface Flow .............................................................................................. 50
TROUBLESHOOTING .............................................................................................................................. 59
APPENDIX: LOADING APPLICATIONS IN BULK ...................................................................................... 60
  Setting Up File Parser Definitions .................................................................................................. 60
  Setting up Constituent Transaction ................................................................................................ 64
  Running the File Parser Process ...................................................................................................... 65
  Reviewing Staged Data .................................................................................................................... 66
  Running the Search Match Post Process ......................................................................................... 68
APPENDIX: DEVELOPERS REFERENCE ................................................................................................... 70
  Application Class Reference ............................................................................................................ 70
CHANGE HISTORY

<table>
<thead>
<tr>
<th>Release</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bundle 21 (April, 2011)</td>
<td>Added Error Handling section (page 57).</td>
</tr>
</tbody>
</table>

This updated guide is available on My Oracle Support as part of the documentation delivered for the releases indicated in the table.
IMPLEMENTATION SUMMARY

This document is aimed for a technical audience who wants to install and configure Admission Applications Web Service (AAWS). AAWS interacts with the following Campus Solutions components:

- User Account Manager
- Constituent Transaction Manager
- Admissions API
- Student Financials Hosted Payment

This document lists the application packages and classes for these components. If you want to customize AAWS, you may need to modify some classes and packages. For instance, if you want to integrate a non PeopleSoft user management system with AAWS, you may need to modify the Campus Identity Manager Application package.

This document also describes how to use the File Parser feature to load applications from a text file into Campus Solutions.

AAWS adheres to open web service standards currently supported in the latest release of Enterprise PeopleTools 8.48+. Therefore, any modern day web service compliant front end (for example, an online application user interface) or third party application can use AAWS.

Note: The AAWS User's Guide describes the various AAWS operations. We recommend that you read the AAWS User's Guide before performing the configuration steps listed in this document.

To use AAWS, complete the following steps:

1. Bundle Installation (Mandatory)

   AAWS is delivered with Campus Solutions Bundle 18 (update ID 782229). Bundle 18 was posted to My Oracle Support in July 2010.

   Minor change to the Counter Setup page was made with Campus Solutions Bundle 19 (update ID 782230). Bundle 19 was posted to My Oracle Support in Oct 2010.

   AAWS Fee integration is delivered with Campus Solutions Bundle 20 (update ID 812865). Bundle 20 was posted to My Oracle Support in Jan 2011.

   Error handling changes were made with Campus Solutions Bundle 21 (update ID 812867). Bundle 21 was posted to My Oracle Support in April 2011.

2. Configuration (Mandatory)
Perform the configuration steps to make AAWS accessible to the user interface.

3. Implement Extensions (Optional)

Design and create the country or institution specific extensions that AAWS or any of the underlying Campus Community components can use.

The following diagram shows an overview of the AAWS architecture:

![AAWS Architecture Diagram]

**Figure 1: AAWS Architecture**

**Service Oriented Architecture User Interface**

Due to AAWS's open standards Service Oriented Architecture (SOA), you can use any technology to develop your user interface, as long as the technology supports web service and SOAP.

The SOA user interface (**SOA UI**) could be an online application that interacts with Campus Solutions through AAWS. Before developing your online application user interface, we recommend that you use a SOAP service tester to become familiar with how the web services processes the admission application data and user information. One way to use a service tester could be by first passing the input parameters to a service operation and then viewing the service operation output.

**Web Services**

The Admission Applications Web Service contains the following web service operations:

- SAD_ADMISISSIONS – Admissions Web Service
- SAD_CREATEAPPL: Create Application
- SAD_GETAPPL: Find Application
- SAD_GETAPPLS: Find Applications
- SAD_SAVEAPPL: Save Application (for later modification or submission)
- SAD_SUBMITAPPL: Submit Application
- SAD_GETATTACH: Get Attachments

- SCC_USERREG - User Account Web Service
  - SCC_USERREG_AUTHENTICATE: Authenticate User
  - SCC_USERREG_CREATEACCT: Create User Account

- SCC_LOV – List of Values Web Service
  - SCC_GET_LOV: Get List Of Values

- SSF_EPAYMENT_TRANSACTION – Hosted Payment Web Service
  - SSF_INITIATE_EPAYMENT: Initiate a Payment
  - SSF_COMPLETE_EPAYMENT: Complete a Payment

**User Account Manager**

The user registration and authentication web service operations interact with *User Account Manager* to register and authenticate the end user. *PeopleTools User Profile*, by default, stores the user account information.

The User Account Manager component contains a number of sub-modules including the Campus Identity Manager and the User Manager. The following diagrams show the core application packages and classes which support the User Account Manager:

![Diagram of Campus Identity Manager](image)

*Figure 2: Campus Identity Manager*
Constituent Transaction Manager handles the admission application and applicant data that AAWS receives and sends to the user interface. When the user interface submits the application and applicant data, AAWS transmits this data to the Constituent Transaction Manager. The Constituent Transaction Manager stores this data in Campus Solutions staging tables.

Constituent Transaction Manager is composed of a number of application classes and PeopleTools components which support the staging and loading of constituent (applicant) and related transaction (application) data into staging tables.

The following diagrams show the application packages and classes in the Constituent Transaction Manager component:
Figure 4: Constituent Transaction Manager (1 of 3)
Figure 5: Constituent Transaction Manager (2 of 3)
Extensions
The Entity Registry feature enables academic institutions to create their own constituent data extensions (that is, you can add entirely new constituent child entities). Refer to the "Extending AAWS" section in this guide for examples.

Admissions API

The Admission Applications Programming Interface (API) moves the application data from the staging tables to the core Recruiting and Admissions tables.

The API supports the creation, modification, and submission of online admission applications. The API is an Object Oriented (OO) based application class hierarchy that represents the admission applications. The class hierarchy contains an Application Manager API that performs key functions with application data. The API also supports the calculation of Application Fees via integration to the Application Fee hosted payment processing.

The following diagram shows the application packages and classes in the Admission Applications Manager component:
Figure 7: Admission Applications Manager

Extensions
The Entity Registry feature enables academic institutions to create their own application data extensions (that is, you can add entirely new application child entities). Refer to the "Extending AAWS" section in this guide for examples.
This chapter describes the setups that you must perform after installing Bundle 18. The chapter includes some setup steps that are listed only for verification purposes and may require little or no setup.

**Verifying Web Service Security Settings**

The SAD_ADMISSIONS, SCC_USERREG, SCC_LOV, SSF_EPAYMENT_TRANSACTION and SCC_SM_SERVICE service operations are delivered with **FULL ACCESS** to the **HCPSERVICE** permission list.

**Note**: SCC_SM_SERVICE supports all aspects of the External Search Match feature and was delivered in Feature Pack 1 as part of the Constituent Web Services (CWS).

To verify whether **HCPSERVICE** permission list is assigned to a service operation with **FULL ACCESS**:

1. Access the General page (**PeopleTools, Integration Broker, Integration Setup, Service Operations, General**).
2. Click the Service Operation Security link. The Web Service Access page appears as shown in this example:

Web Service Access

Service: SAD_ADMISSIONS
Operation: SAD_CREATEAPPL

<table>
<thead>
<tr>
<th>Permission</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCS_PP_SERVICE</td>
<td>Full Access</td>
</tr>
</tbody>
</table>

Figure 8: Example of a General page

Figure 9: Example of a Web Service Access page
Setting Up User Accounts

In PeopleTools 8.48+, any anonymous inbound request originates from the ANONYMOUS node. For example, when an applicant requests for a user ID, the user registration service operation associates the request with the ANONYMOUS node.

To enable anonymous access to Campus Solutions web services in a secure fashion, the PeopleSoft system, by default, associates all anonymous requests with a default account. You will need to set up a default account that has minimal system access. This default account is the SCC_GUEST User Profile.

To create the SCC_GUEST account, use the User Profiles component (PeopleTools, Security, User Profiles, User Profiles).

This example shows how the General page of the User Profiles component is set up for SCC_GUEST:

![Figure 10: Example of a General page (SCC_GUEST)](image1)

This example shows the ID page for SCC_GUEST:

![Figure 11: Example of an ID page (SCC_GUEST)](image2)
This example shows the Roles page for SCC_GUEST:

![Figure 12: Example of a Roles page (SCC_GUEST)](image)

After successfully processing the user registration request, the user registration service operation creates a new account for the user. In other words, the service operation creates a PeopleTools user profile for the user. The service operation uses the template account SCC_SS_TEMPLATE to create this new account.

You will need to create the SCC_SS_TEMPLATE account. **Note:** Assign this account any common defaults that your institution wants new users to automatically receive when registering.

To create the SCC_SS_TEMPLATE account, use the User Profiles component (**PeopleTools, Security, User Profiles, User Profiles**).

This example shows how the General page of the User Profiles component is set up for SCC_SS_TEMPLATE:

![Figure 13: Example of a General page (SCC_SS_TEMPLATE)](image)

This example shows the ID page for SSC_SS_TEMPLATE:
Performing a Security Analysis

The example screenshots in the previous section related to SCC_SS_TEMPLATE role assignment and configuration are for use only in a testing or demo environment. Before you migrate AAWS and the other services to a production environment, we recommend that you complete a thorough analysis of your institution's security requirements.

Also, we recommend that you allocate the SCC_SS_TEMPLATE user only a minimal amount of system access that is required to execute the web services. This access includes authorization to access AAWS, user account, search match, and list of values web services and the underlying HCM component interfaces that are required to operate these services. To define this access, any role that you create should have access to these web services, along with at least the following component interfaces (which are used by AAWS to access the HCM Person data):

- HCR_EMPLOYEE_CHECKLIST_SRV
- HCR_IDENTIFICATN_DATA_SRV
• HCR_JOB_DATA_POI_SRV
• HCR_MIL_EE_TRK_SRV
• HCR_NAMES_OTHER_SRV
• HCR_PERSONAL_DATA_SRV

Publishing Web Services

Use the Provide Web Service page to publish the following web services (PeopleTools, Integration Broker, Web services, Provide Web Service).

• SAD_ADMISSIONS
• SCC_USERREG
• SCC_LOV
• SSF_EPAYMENT_TRANSACTION

**Optional: If you want the system to perform a search for potential duplicate records within an external system when a user submits an online application, ensure that the CWS External Search Match is set up. Refer to the "External Search/Match" chapter in the PeopleSoft Enterprise Campus Community Fundamentals 9.0 PeopleBook for more information. If you are not using an external data hub to store applicant records, you do not need to set up CWS External Search Match.

Refreshing the HCM Registry Cache

A new version (1.1) of the Campus Solutions AddPersonCS service has been delivered. This may require the HCM service registry to be refreshed.

To refresh the HCM service registry:

1. Access the HCM Interface Registry page (Set Up HRMS, System Administration, HCM Registry, Service Registry).

Figure 16: HCM Interface Registry page
2. Click the **Refresh Cache** button.

### Setting Up Integration Broker

**Note:** We assume that you have performed and validated all the basic Integration Broker setups. Refer to the Integration Broker PeopleBooks for information on Integration Broker gateway setup and service configuration.

In PeopleTools 8.48+, any anonymous inbound request originates from the **ANONYMOUS** node. The PeopleSoft system, by default, associates all anonymous requests with the **SCC_GUEST** user profile. For information on setting up the **SCC_GUEST** user profile, refer to the "Setting Up User Accounts" section.

Perform the following steps to update the **ANONYMOUS** Integration Broker node:

1. Access the Node Definitions page (**PeopleTools, Integration Broker, Integration Setup, Service Operations**).
2. Ensure that Node Type is set to **External**, the Default User ID is set to **SCC_GUEST**, and the Active Node and Segment Aware options are selected:

   ![Node Definitions page](image)

   **Figure 17: Node Definitions page**

3. Enable WS-Security for this node by selecting an authentication token type:
Verifying Campus Solutions SOA Framework Setup Entries

During installation, the system automatically inserts configuration data into the Campus Solutions SOA Framework setup tables.

To verify whether the setup entries exist:

1. Access the Request Handlers component (*Set Up SACR, System Administration, Integrations, Request Handlers*).
2. For all the SAD_ADMISSIONS web service operations, ensure that the fields on the Request Handlers page contain the following values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package Name</td>
<td>SCC_OLA</td>
</tr>
<tr>
<td>Path</td>
<td>Handlers</td>
</tr>
<tr>
<td>Application Class ID</td>
<td>Admissions</td>
</tr>
</tbody>
</table>

3. For all the SCC_USERREG web service operations, ensure that the fields on the Request Handlers page contain the following values:
4. For all the SCC_LOV web service operations, ensure that the fields on the Request Handlers page contain the following values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package Name</td>
<td>SCC_COMMON</td>
</tr>
<tr>
<td>Path</td>
<td>Handlers</td>
</tr>
<tr>
<td>Application Class ID</td>
<td>LOVHandler</td>
</tr>
</tbody>
</table>

5. For all the SSF_EPAYMENT_TRANSACTION web service operations, ensure that the fields on the Request Handlers page contain the following values:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package Name</td>
<td>SSF_EPAYMENT_TRANS</td>
</tr>
<tr>
<td>Path</td>
<td>HANDLERS</td>
</tr>
<tr>
<td>Application Class ID</td>
<td>Payments</td>
</tr>
</tbody>
</table>

This example shows the Request Handlers page for a service operation with the correct setup entries:
**Setting Up Logging**

This is an optional setup.

Use the Logging page to enable and configure the logging feature (*Set Up SACR, System Administration, Integrations, Logging*).

**Figure 20: Logging page**
Note: You should **not** enable logging in a production environment. SOA framework logging is only appropriate for a testing, demo or development environment or when performing critical troubleshooting activities.

**Verifying Constituent Transaction Manager Setup Entries**

During installation, the system automatically installs several Transaction Manager Configuration entries. This includes sample Transaction Manager Configuration entries for the New User Registration and the Admission Applications transactions. The sample configuration is shown in the following screen shots.

For information on adding new transactions, refer to the *Managing PeopleSoft Admission Transactions* PeopleBook chapter.

To verify whether the sample configuration exists in your system, access the Transaction Setup component (Set Up SACR, System Administration, Utilities, Constituent Transaction Mgmt, Transaction Setup). Compare the page settings on your system with the following examples.

This shows the delivered Transaction Setup page configuration for the New User Registration transaction:
Figure 21: Delivered Configuration for New User Registration (Transaction Setup page)

Note that the system uses the Transaction Setup page to identify the New User Registration transaction.

This shows the delivered Transaction Setup page configuration for the Admission Applications transaction:

<table>
<thead>
<tr>
<th>Transaction</th>
<th>ADMISSIONS_REGISTRATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction Name</td>
<td>ADMISSIONS_REGISTRATION</td>
</tr>
<tr>
<td>Transaction Description</td>
<td>NUR Provisioning</td>
</tr>
<tr>
<td>Transaction Status</td>
<td>Active</td>
</tr>
<tr>
<td>Online Transaction</td>
<td>Yes</td>
</tr>
<tr>
<td>New User Registration</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Transaction Handler

<table>
<thead>
<tr>
<th>Root Package ID</th>
<th>SCC_OLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>TRANSACTION</td>
</tr>
<tr>
<td>Application Class ID</td>
<td>DefaultTransaction</td>
</tr>
</tbody>
</table>

### Transaction Status

<table>
<thead>
<tr>
<th>Record Name</th>
<th>SCC_STG_CONSTN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Name</td>
<td>SCC_STG_STATUS</td>
</tr>
<tr>
<td>Date Field Name</td>
<td>SCC_STG_STS_DT</td>
</tr>
</tbody>
</table>

### Constituent Handler

<table>
<thead>
<tr>
<th>Root Package ID</th>
<th>SCC_BL_TRANSACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>INTFC</td>
</tr>
<tr>
<td>Application Class ID</td>
<td>Default Constituent</td>
</tr>
</tbody>
</table>

### Partition Data

- By Constituent
- By Transaction
Figure 22: Delivered Configuration for Admission Applications (Transaction Setup page)

Note that:

- The system uses the Transaction Setup page to identify the Admission Applications transaction.
• In the delivered configuration, the system has indicated that the search match process runs in *Realtime* when the application is *submitted*.

This shows the delivered Search/Match Setup page configuration for the Admission Applications transaction:

![Figure 23: Delivered Configuration for Admission Applications (Search/Match Setup page)](image)

Note that you can change the search match parameters based on your requirements.

**Configuring AAWS**

This section discusses how to set up:

- Attachments URL
- AAWS Applications
- Application Extensions
- Data Update Rules
- List of Values

**Setting Up the Attachments URL**

Use the URL Maintenance page to define the location that AAWS uses to store application attachments. To navigate to the URL Maintenance page, select *PeopleTools, Utilities, Administration, URLs*.

In this example, attachments are stored in a FTP Server:
Setting Up AAWS Applications

The Application Configuration component is the main AAWS configuration component. To navigate to the Application Configuration component, select Set Up SACR, Product Related, Recruiting and Admissions, Application Configuration, Application Configuration.

This component has four tabs that are shown in the following screen shots. This section is a brief introduction to the typical configuration options available for AAWS. The configuration presented in the screen shots is minimal and you should adjust the configuration according to your institutions needs. Please see the AAWS User’s Guide and the Managing PeopleSoft Admission Transactions PeopleBook chapter for further configuration guidance.

Tab 1 – Appl Config (Transaction Mapping)

When a user submits an application through AAWS, the Campus Solutions system uses the Appl Config page to associate the application with a transaction. The system uses this association to run search/match and process the application data according to the Transaction Manager setup.

In this example, various academic careers are mapped on the Appl Config page:

![Figure 25: Appl Config page](image-url)
**Tab 2 – Application Setup (Application Center Defaulting)**

When a user submits an application through AAWS, the system uses the Application Setup page to assign an application center to the application. Each application must have an application center.

This is an example of the Application Setup page:

![Application Setup page](image)

**Figure 26: Application Setup page**

**Tab 3 – Prospect Setup (Prospect Creation)**

The system uses the Prospect Setup page to create prospect data when the user saves or submits the online application. This is an optional setup. If you do not want to create a prospect record when the application is saved or submitted, leave the page blank.

This is an example of the Prospect Setup page:

![Prospect Setup page](image)

**Figure 27: Prospect Setup page**

**Tab 4 – School Setup (Unlisted Organizations and Home Schooling)**

Use the School Setup page to enter optional processing defaults for Unlisted or Home Schooling organizations.

This is an example of the School Setup page:

![School Setup page](image)
The Require Manual Reconciliation option can be used as an alternative to supplying a single default. When this option is chosen and an application contains school data with an Unlisted or Home School, then the application itself is not immediately submitted but instead is delayed in the staging tables until such time as an Administrator can manually review the application and resolve the Unknown Organization via the Staging Review Components.

**Setting Up Application Extensions**

Use the Entity Registry page to configure AAWS extensions (Set Up SACR, System Administration, Entity, Entity Registry). Refer to the "Extending AAWS" section in this document for more information.

**Setting Up Data Update Rules**

This is an optional setup. Use the Data Update Rule Entry page to associate data entities with data update rules. This page enables you to configure whether or how AAWS can update data in the admissions and person tables. To navigate to the Data Update Rule Entry page, select Set Up SACR, System Administration, Utilities, Constituent Transaction Mgmt, Data Update Rule Entry.

**Setting Up List of Values**

List of Values configuration is optional but recommended. The degree of configuration required is entirely dependent on the requirements of your institution. The List of Values Web Service will perform out-of-the-box with no configuration for simple scenarios. To navigate to the setup page, select Set Up SACR, Product Related, Recruiting and Admissions, Application Configuration, List Of Values.
Setting Up Hosted Payment

Application Fees are optionally required during application submission as configured in the Application Center Setup page. If fees are required to be paid then a prerequisite is to set up the hosted payment adapter and merchant which will facilitate the online payment.

Firstly, create a new HTML object in Application Designer. The HTML object is used to store UI specific HTML and JavaScript which will direct the Self Service user interface during the payment integration flow. Please see the User Interface Considerations section for detailed technical information on what this HTML object may need to contain.

Next, set up a new hosted payment adapter for AAWS pointing to the same application class as the one used for PIA Self Service. This step is required because the HTML definition (created above) is attached to the Adapter itself.

Setup SARC>System Administration>Utilities>Adapter Registry
Figure 30: Hosted Payment Adapter Setup page

Next, attach the new HTML definition to the Hosted Payment Adapter settings.

Setup SACR>Common Definitions>Electronic Payments>Hosted Payment Adapter Setting

Figure 31: Hosted Payment Adapter Settings Setup page Hosted Payment Adapter Tab

The Hosted Payment Adapter Settings Parameters tab should be set up according to your Third Party Payment Provider’s requirements.
Figure 32: Hosted Payment Adapter Settings Setup page Parameters Tab

Next, the hosted payment adapter is attached to a Payment Merchant. If you are already using the hosted payment framework for Student Financials’ Self Service or Cashiering, you will still need to set up new Payment Merchants with a different set of Third Party Merchant IDs. Your third party will require return URLs in order to navigate the user back to the online application. These return URLs are typically tied to the Third Party Merchant ID defined in the Payment Merchant.

Note that the hosted payment adapters as delivered in Feature Pack 2 (Bundle 16) use the Get Token (91) and Authorize Only (92) transactions defined in the Electronic Payment Setup section of the Payment Merchant setup. If you plan to use these adapters as delivered, you will need to set up this section in the same manner as defined in the Electronic Payment Integration Developer’s Reference Guide. In addition, you will need to set up the corresponding Integration Broker Related objects for these transactions. Please refer to the Electronic Payment Integration Developer’s Reference Guide for more details.

You are required to configure one for each payment method you accept (credit card or electronic check).

Setup SACR>Common Definitions>Electronic Payments>Payment Merchant
Finally, Payment merchants are attached to SF Merchants which are associated with the Admissions Application Center.

Credit transactions (Credit Only – 93 as defined above) can only be generated for transactions posted to the student account. Since payments received through the online admissions application are posted to the General Ledger directly, credit transactions are not supported for these application fee payments.

*Setup SACR > Common Definitions > Self Service > Student Financials > SF Merchants*
Figure 34: SF Merchant Setup page

Note the convenience fees are not currently supported for online application fee payment.
EXTENDING AAWS

This chapter covers how to add new entities or extend existing entities.

**Scenario** – An academic institution has a requirement to add one or more new entities into the application process. The new entity could represent entirely new data or it could represent an extension to one of the existing delivered entity data types. This may involve either or both application and constituent data. You can use the Entity Registry feature for all these situations.

Once you register an entity, the entity and its data can flow seamlessly in and out of the Campus Solutions system (the marshalling of a database record into an XML structure occurs automatically and is driven by the PeopleTools metadata).

**To create an entity:**

1. Create entity database tables (staging and production versions).

2. Create the supporting entity application class. Alternatively, create a placeholder for the application class, then use the Entity Registry page to generate the class code, and finally copy the generated code into the placeholder location.

3. Use the Entity Registry page to register the entity application class.

All AAWS entities are represented by the IEntity interface SAD_ADM_APPL:Accessors:IEntity. The IEntity interface defines the core behavior that any participating entity needs to support in order to be managed by AAWS. For each new distinct entity that you want to create for AAWS, an application class implementing the IEntity interface needs to be created and registered.

Example:

**Step 1**: Create entity tables:

This example extension record, SCC_EXTN, contains a map of sample questions and answers that need to be captured during the application submission process.

This is the *production* table for the entity:

<table>
<thead>
<tr>
<th>Num</th>
<th>Field Name</th>
<th>Type</th>
<th>Key</th>
<th>Ord</th>
<th>Del</th>
<th>Key</th>
<th>Del</th>
<th>Src</th>
<th>Del</th>
<th>List</th>
<th>Sys</th>
<th>Audit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EMPID</td>
<td>Char</td>
<td>Key</td>
<td>1</td>
<td>Asc</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>ADM_APPL_NBR</td>
<td>Char</td>
<td>Key</td>
<td>2</td>
<td>Asc</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>SEGNO</td>
<td>Nbr</td>
<td>Key</td>
<td>3</td>
<td>Asc</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>HINT_QUESTION</td>
<td>Char</td>
<td></td>
<td></td>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>QUESTION_ANSWER</td>
<td>Char</td>
<td></td>
<td></td>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

*Figure 35: Example of Production Table Fields*
In this example, the new extension table is a child entity to the top level Application entity. Consequently, the compound key is **EMPLID, ADM_APPL_NBR** and then zero, one or more additional low order keys. Here we are using **SEQNO** to identify any number of Q&A entries for the individual application.

The **Staging** table for this entity will hold the staged entity data before it is promoted to the production table. The keys for this table are similar to the staging table except that the high level key is **SCC_TEMP_ID**.

---

**Figure 36**: Example of Staging Table Fields

**Step 2**: Create a new entity application class:

Create a new application class that extends the **SCC_COMMON:ENTITY:StagedEntity** class. For this example, we have created the **SAD_ADM_APPL:Accessors:QuestionAnswerExt** class.

---

The PeopleCode for **SAD_ADM_APPL:Accessors:QuestionAnswerExt** is delivered with Bundle 18. The significant aspects of the new QuestionAnswerExt entity application class are highlighted here.

**The class is extending SCC_COMMON:ENTITY:StagedEntity:**

```java
class QuestionAnswerExt extends SCC_COMMON:ENTITY:StagedEntity
```
Any new class that you create must declare all its own immutable keys and mutable properties:

/* Immutable properties - Additional low order Keys */
property string ADM_APPL_NBR get set;
property number SEQNO get set;
/* Mutable Properties */
property string DESCR100 get set;
property string QUESTION_ANSWER get set;

Any new class that you create may choose to override the validate() method in order to validate data prior to saving:

method validate
    /* Do nothing */
    End-If;
end-method;

Any new class that you create may choose to override the preSave() method in order to default fields before the user saves the application:

method preSave
    If Value(%This.ADM_APPL_NBR) = 0 Then
        %This.ADM_APPL_NBR = (%This.getParent() As
        SAD_ADM_APPL:Accessors:Application).ADM_APPL_NBR;
        %This.data.ADM_APPL_NBR.Value = %This.ADM_APPL_NBR;
    End-If;
end-method;

Any new class must declare get/set methods for all of its properties:
get DESCR100
   /* Returns String */
   Return %This.data.DESCR100.Value;
end-get;

set DESCR100
   /* &NewValue as String */
   %This.data.DESCR100.Value = &NewValue;
end-set;

get QUESTION_ANSWER
   /* Returns String */
   Return %This.data.QUESTION_ANSWER.Value;
end-get;

set QUESTION_ANSWER
   /* &NewValue as String */
   %This.data.QUESTION_ANSWER.Value = &NewValue;
end-set;

get ADM_APPL_NBR
   /* Returns String */
   Return %This.data.ADM_APPL_NBR.Value;
end-get;

set ADM_APPL_NBR
   /* &NewValue as String */
   %This.data.ADM_APPL_NBR.Value = &NewValue;
end-set;

get SEQNO
   /* Returns Number */
   Return %This.data.SEQNO.Value;

end-get;

set SEQNO
   /* &NewValue as Number */
   %This.data.SEQNO.Value = &NewValue;
end-set;

Note: You can use the Entity Registry page to generate the PeopleCode for the application class. Refer to Step 3 for the discussion on the code generation feature.

Notes:

- Entity key values are stored in the entity immutable properties.
- Entity key values are also directly stored in the underlying entity record.
- Entity non-key values are set via the get/set accessor methods.

This approach ensures integrity of the contained entity data via encapsulation.
**Step 3:** Register the Entity Application class with Entity Registry:

Access the Entity Registry page (*Set Up SACR, System Administration, Entity, Entity Registry*).

When creating a new entity, it will give you this page.

![Entity Registry](image)

**Figure 38:** Entity Registry page

Fill out the following fields:

- **Name (Required)** – A descriptive name of the entity.
- **Status (Required)** – Set it to active, as inactive makes the entity, and any children, effectively ignored.
- **Description** – A description of what the entity is used for.
- **AppClass (Required)** – This should be the empty application class created in the previous step, and it will contain the entity code.
- **Prod Record (Required)** – The production record created in step 1.
Stage Record (Optional) – While this field is optional, in the context of AAWS, it should be populated with the stage record created in step 1.

Element Name (Required) – The element name to be used for this entity in the xml; it should not match any other entity’s element name and cannot contain spaces.

Ignore the Apply Data Update Rule or anything within the Children grid, unless you intend for this entity to have children. Adding a child will be covered in a later step.

Save the entity.

**Entity Registry**

Press the Generate Code button and you will get the following page.

---

**Figure 39: An Example Entity Registry page**

Press the Generate Code button and you will get the following page.
Field Descriptions

- **Base AppClass** (Required) – Specify the base application class for this entity. There are three choices.
  - Basic Entity – A class for entities that do not need to stage data, for save and fill purposes it is assumed direct access to the production record is allowed. It is not recommended to use this entity for any extensions to AAWS Constituent or Application, since both of these are staged.
  - Staged Entity – A base class for entities that stage data where direct access to the stage and production records are allowed from the entity. This is likely the entity type to be used when implementing AAWS extensions.
  - Staged HR Entity – A base class for entities that stage data where direct access to stage records is allowed, but access to production data is performed via HR services. This is used internally for access to HR data; it can also be used for extensions that store data in different or partitioned areas outside of the Campus Solutions database.

- **Grid** – This grid shows every property that represents a field in a record. The record definitions are scanned for both the stage and production records to build the grid.
  - Property Type – The data type determined for the property based on the field type on the record.
  - Field Name – The name of the field, this will also be the name of the property.
In Production – If the field exists in the production record.

In Stage – If the field exists in the stage record.

Ignore – Specifies if you want this field ignored for the purposes of xml. This means the field will not show up in the schema and will not be parsed if it is in incoming xml and will not be sent if it is in outgoing xml. This is useful if the field is entirely managed internally (for example, an internal use only system identifier or counter field). There are a few fields that are ignored by default and cannot be set to un-ignored (i.e. SCC_ROW_ADD_OPRID, SCC_ROW_ADD_DTTM, SCC_ROW_UPD_OPRID, SCC_ROW_UPD_DTTM). When the code is generated, this means the field will be added to the array &ignoreFields.

- Custom Pre-Save code – Adds the methods preSave and preSaveFirst to the generated code. These methods are called as part of the preSave process.
  - preSave – This is called prior to the save process to do any final data processing.
  - preSaveFirst – This is called once per childEntity before preSave. This is useful if you have code that needs to be run across all child entities of a particular type, or you need to remove entities prior to saving.

- Use Custom Validation – Adds the methods Validate and validateFirst to the generated Code.
  - Validate – Runs as part of the validation process for this entity. Generally, this method should call commonValidate, which is responsible for field level validations. Any custom validation can also be added.
  - validateFirst – This is called once per childEntity, before any other validation. This method is good for running validations that may need to compare entities against each other that you do not need to run during the validation for every entity.

- Set Default – Adds an override for setDefault. Normally, setDefault is based on the field defaults specified on the record; it can be overridden, though, for custom default setting. It is recommended to call %Super.setDefault() first to set the default field values and then set any special overrides.

Once you have decided on the specific options, press Generate Code. This will provide the code so you can cut and paste that code into application designer.

Notes:

- The Registry records the production and staging record names.

- The entity data is contained within the nominated extension SCC_EXTN tag and is a child to the Application entity.

- To use the Generate Code feature, simply enter production and staging record names and click the button. You will then be prompted to select a base class, the fields to include and which method stubs you require as shown in this example:
Figure 41: Example Entity Code Generation page

When you click the Generate Code button on the Entity Code Generation page, the system displays the sample code template:

```
import SCC_COMMON;ENTITY:StagedEntity;
import SCC_COMMON;ENTITY:Entity;
import SCC_SL_TRANSACTION;INTFC:MessageLog;

class QuestionAnswerExt extends StagedEntity
method QuestionAnswerExt(&p_parent As IEntity);
method Presave();
method PresaveFirst();
method validate(&p_messageLog As SCC_SL_TRANSACTION;INTFC:MessageLog out);
method validateFirst(&p_messageLog As SCC_SL_TRANSACTION;INTFC:MessageLog out);
```

Figure 42: Entity Code Generation page – Example Generated Appclass Code

The entity can now be added as a child to a currently existing entity.

Open the entity you want to add your newly created entity as a child of. The grid is located at the bottom of the page as shown below:
Figure 43: Entity Registry Children Grid

- **Order** – Specify the sequence of these child entities in the XML schema.
- **Entity Name (Required)** – The Child Entity, prompting off a list of all entities.
- **Status** – Displays the current status of the entity.
- **Embed** – Specify if the entity data should be embedded directly in the parent. This is useful for certain sibling record purposes where you do not want the XML to show the child at a separate level from the parent.
  - Non-Embedded
    
    ```xml
    <Entity>
    <Data1/>
    <Data2/>
    <Entity2>
    <Data3/>
    </Entity2>
    </Entity>
    ```
  - Embedded
    
    ```xml
    <Entity>
    <Data1/>
    <Data2/>
    <Data3/>
    </Entity>
    ```
- **Min** – The minimum number of occurrences of this entity under the parent, which can be any number from 0 on up.
- **Max** – The maximum number of occurrences of this entity under the parent. The number 0 means the maximum is unbounded, or any number greater than the min can be used.
- **Element Wrapper** – If a value is specified here, it is used as the tag to wrap all the child occurrences of this entity under the parent.
  - Without Wrapper
    
    ```xml
    <Entity>
    <Data1/>
    <Data2/>
    <Entity2>
    <Data3/>
    </Entity2>
    ```
• View – Opens the entity registry of the child entity in that row.

To be clear, at this point the WSDL will not reflect the new entity. However, if the entity is added to an incoming message, and no schema validation is occurring, the incoming message containing this data will be processed and this entity will be handled.

As an example, the Q&A Application Extension entity is registered as a child entity to the Application entity. The Q&A Application Extension entity appears in the EXTENSION_DATA section of the outbound XML response message. This section appears within the core Application ADM_APPL_DATA section of both inbound and outbound messages. Note that the choice of tag names (for example, EXTENSION_DATA) is configurable.
Figure 44: Q&A Application Extension as a child entity for Application

The following is an example of the XML message that contains the Q&A Application Extension entity data:

```xml
<ADM_APPL_DATA class="R">
    ...
    <ADM_APP_CAR_SEQ class="R">
        <ADM_APPL_PROG class="R">
            ...
            <ADM_APPL_PLAN class="R">
                ...
            </ADM_APPL_PLAN>
        </ADM_APPL_PROG>
    </ADM_APP_CAR_SEQ>
    <EXTENSION_DATA>
        ...
    </EXTENSION_DATA>
</ADM_APPL_DATA>
```
Press Generate XSD on the root entity (constituent or application) you have modified. This will provide a schema that you can copy and paste into the messages to modify the overall message schema.

**Entity Schema**

![Entity Schema](image)

**Figure 45: Q&A Application Extension Generated Code**

Once an extension has been added to the registry it needs to be reflected in the XML schema definition for either the Constituent or Application entity.

Copy the schema generated for the Application entity to the schema in Integration Broker message SCC_ENTITY_APPLICATION.

Open PeopleTools > Integration Broker > Integration Setup > Messages. Search for message SCC_ENTITY_APPLICATION and select tab “Schema”. 
Copy the schema generated for the Constituent entity to the schema in message SCC_ENTITY_CONSTITUENT.

Open PeopleTools > Integration Broker > Integration Setup > Messages. Search for message SCC_ENTITY_CONSTITUENT and select tab “Schema”.

Press Edit Schema and paste in the generated schema, then press Save. Once the schemas have been updated and the service WSDL re-published, any newly added extensions will be recognized.
It is beyond the scope of this guide to cover the best practices for user interface design and an effective online application user experience. However, this chapter presents some key considerations for designing and creating a user interface that interacts with AAWS.

In addition to user interfaces interacting with AAWS, a third party application can also be interacting with AAWS by passing in completed application data to AAWS for further processing.

We recommend academic institutions to use Oracle Application Development Framework (Oracle ADF) to develop an online application user interface. However, academic institutions can use any technology or tool that complies with the minimal technical prerequisites specified in the AAWS User's Guide. Any third party application technology deployed in your institution can be used to communicate with AAWS as long as it complies with the specified prerequisites.

In this chapter, we will examine a typical data collection flow for an online application user interface. This will reveal several recommendations on how a user interface can communicate with AAWS. The following figure depicts a typical data collection flow for an online application user interface:
Discussion of the User Interface Flow

Start: Unregistered User
This is the starting point for an online application user interface.

Essentially all applicant users start in this state. Some users may already have a username and password to access the online application (these users are not required to complete the new user registration process).

The PeopleSoft system considers any user at this point of time as anonymous. Therefore, these anonymous users receive minimal access rights to AAWS until they either log into the system or complete the new user registration process.

New User Registration

Any (anonymous) user who needs an account (username/password combination) in order to access the online application must complete the new user registration process.

A typical self service online application UI might perform the following actions:

- Present a “Register” hyperlink that the user can click to initiate the new user registration process.
- Present a New User Registration page to capture basic information from the user. This might typically include:
  - Name
  - Address
  - Email Address
  - Username
  - Password
  - Any constituent-related information that is of interest to the institution. This can be core constituent information or an institution extension as defined in the Entity Registry.
- Validate that required information has been entered by the user.
- Formulate a request message from the constituent information.
- Submit the request message to the SCC_USERREG_CREATEACCT service operation.
- Present the user with any error response messages from AAWS and allow the user to correct the submitted data.

LOGON (Authentication)

Logon is the process in which the institution's Identity Management or User Profile Management system authenticates the username and password. This authentication is not a one-time event. Re-authentication must occur on every request that the online application sends to AAWS. This is a security requirement for the web service architecture and is enforced by PeopleSoft Integration Broker as a standard behavior.

Academic institutions can use the delivered SCC_USERREG_AUTHENTICATE service operation to validate a user's credentials before the user can access the online application. Typically you want to ensure that a user has the key to the door before allowing them to proceed. This is the purpose of the SCC_USERREG_AUTHENTICATE service operation.

Once the system has authenticated the user, it is also an appropriate time for the user interface to perform any initialization or setup to prepare the user for entry into the online application. Such setup may include requesting AAWS for a bulk list of values data that the user interface can subsequently
display to the user. Performing this activity up front may result in a general performance improvement and improved user experience.

A typical self service online application UI might perform the following actions:

- Present a “Login” hyperlink that the user can click to initiate the user authentication process.
- Present a Login page where the user can enter a username and password.
- Validate that the required information has been entered.
- Formulate a request message from the login information.
- Submit the request message to the SCC_USERREG_AUTHENTICATE service operation.
- Present the user with any error response messages from AAWS and allow the user to correct the submitted data.
- Once a successful login response is received, the user interface should maintain the credentials within the program state and supply these credentials on every subsequent request to AAWS as part of the WS-Security headers (thereby enabling ongoing authentication).
- The academic institution's Identity Management or User Account Management system will have its own configured rules surrounding login retries.
- Optionally, formulate a bulk List-Of-Values request message.
- Submit the request message to the SCC_GET_LOV service operation.
- Store the List-Of-Value results in a temporary storage area for use throughout the online application functioning.

Get Application Summary

Once the user has been authenticated, it is likely that the user would then expect to see some form of online application launch page. It is also likely that returning users would expect to view or update any applications they have saved or submitted previously. To implement this kind of scenario, the online application user interface can request from AAWS the application summary information for the logged in user and then display this information on the launch page.

For first time users, the online application does not display saved or submitted application information. For these users, the online application can display options for creating a new application or for logging out.

A typical self service online application UI might perform the following actions:

- Formulate a request message.
- Submit the request message to the SAD_GETAPPLS service operation.
- Receive the application summary response message from AAWS.
- Based on the application status, present each application to the user in a grid style format for subsequent actions.
- Typically, applications which are in a Submitted status are not eligible for update and a user interface would need to enforce this rule depending on the institution's requirements.

Create Application

From the online application launch page, a user may choose to start the application process by creating a new application.

A typical self service online application UI might perform the following actions:
• Present an “Apply” hyperlink that the user can click to initiate the application process.
• Present an initial Academic Learning Area page where the user can choose a field of study. For example, on this page the user supplies the required values to the SAD_CREATEAPPL service operation. The required values that the user enters might include academic program, plan, and sub plan.
• Formulate a request message.
• Submit the request message to the SAD_CREATEAPPL service operation.
• Receive one or more admission application templates with default values from AAWS.
• Present the one or more applications with the default values to the user through one or more user interface pages.
• Allow the user to enter in all required application details.

Get Fee Configuration

Application fees while optional are likely a common scenario that needs to be supported by the UI. At some point after application creation (where the Application Center assigned to the application will be known to the UI) but prior to application submission (when fees may be payable) the UI needs to obtain the Application Fee configuration options. The fee options dictate what fee payment preferences should be presented to the self service applicant by the UI. Options surrounding fee payment include:

1. Whether an application fee is required
2. Whether the applicant is given the option to “Pay later”.
3. Whether the applicant can be given the option to request a “Waiver” for the fee.

Based on what fee options are configured the UI needs to react appropriately and present the applicant with only the valid options and this will in some cases alter the UI page flow presented to the applicant.

A typical self service online application UI might perform the following actions:

• Collect the Academic Learning Area preferences that the applicant enters during the Application creation step.
• Use the preferences to create a request message for the List of Values Web Service, the request must include a look up of:
  1. The SAD_LISTVAL_VW record to retrieve the Application Fee options.
  2. The SSF_ADM_TND_VW record to retrieve the tendering options.
• Submit the request message to the SCC_LOV service.
• Receive the Application Fee configuration options.
• Act on the Application Fee configuration Options in the following way:
  o If the “Pay Later” option is activated, then present the applicant with the choice to pay later (the applicants choice is then used in the later Submit Application step).
  o Store all Application Fee configuration and Tendering configuration options in UI storage for use in later Present Fee and Options step.

Update Application

The online application launch page may show applications that the user has previously saved or submitted. The user may wish to submit a previously saved application.

A typical self service online application UI might perform the following actions:

• Present an “Update” hyperlink that the user can click to initiate the application update process.
• Formulate a request message.
• Submit the request message to the SAD_GETAPPL service operation.
• Receive an in-process or draft application from AAWS.
• Present the application to the user through one or more user interface pages.
• Allow the user to modify or enter the application details.

Edit Application

This activity represents the user completing details on one or more pages in the online application. This activity mostly involves the user interface presenting the page flow and collecting data from the user. If your institution has implemented attachments, then the returning user may wish to view any existing attachments associated with an application.

When the user edits the application data, the online application user interface may present dynamically generated prompts and dropdown list boxes. The online application may need to populate the values for these prompts and list boxes through calls to the List-Of-Values service. Dynamic prompting occurs in situations where the system uses the user input to display the user interface dynamically. An example of dynamic prompting is when the user selects an undergraduate career; the Program field on the online application displays only undergraduate programs for the user to select.

A typical self service online application UI might perform the following actions:

• Present a series of pages for the user to interact with.
• Optionally, formulate a dynamic List-Of-Values request message.
• Submit the request message to the SCC_GET_LOV service operation.
• Present the List-Of-Value results in the online application page fields.
• Optionally, formulate an attachment request message.
• Submit the request message to the SAD_GET_ATTACH service operation.
• Decode the response from AAWS and present the binary content of the attachment in a suitable viewer.

Save Application

When the user is creating or updating an application, he or she may want to pause and save the application for later modification or submission. The saved application is an in-process or draft application and AAWS places this draft application in a staging area. Subsequently, the user may submit this application or the system administrator may delete this application.

A typical self service online application UI might perform the following actions:

• Present a “Save For Later” hyperlink on any page in the page sequence. Generally, any page after the user starts creating the application may display this link. The user clicks this link to save the application.
• Prompt the user that the application is saved and the academic institution might use the applicant information for marketing purposes (if prospect creation is configured to occur on save).
• Formulate a request message.
• Submit the request message to the SAD_SAVEAPPL service operation.
• Receive and process the response from AAWS:
  o On a Success response there are various options:
    1. Present the user with a confirmation page.
    2. Take the user back to the Get Application Summary page.
    3. Log out the user from the online application.
  o On an Error response:
1. Based on the error codes that AAWS returns, position the user at an appropriate page of the user interface to allow data correction.
2. Present validation error messages to the user.
3. Present system error messages to the user.

Submit Application

After a user completes the application, the user interface should present a Submit Application option. Also, at submission time, the interface can present any number of disclaimer pages to the user. Generally, after a user submits the application, the user cannot update the application and the user interface would need to enforce this rule using the application status value. The user interface can retrieve this value using the SAD_GETAPPLS service operation.

A typical self service online application UI might perform the following actions:

- Present a “Submit” button on the final online application page.
- Prompt the user with any required disclaimers.
- Formulate a request message. Note that if a fee is due and the applicant has opted to “Pay Later” during the Get Fee Configuration step then this intent must be indicated in the request message.
- Submit the request message to the SAD_SUBMITAPPL service operation.
- Receive and process a response from AAWS. There are three alternative responses that the UI needs to cater for:
  - A Success response and Fee Payment is NOT required:
    1. Present the user with a confirmation page.
    2. Take the user back to the Get Application Summary page.
    3. Log out the user from the online application.
  - A Success response and Fee Payment IS required:
    1. Extract the fee amount and currency code from the response.
    2. Proceed to the Present Fee and Options page.
  - An Error response:
    1. Based on the error codes that AAWS returns, position the user at an appropriate page of the user interface to allow data correction.
    2. Present validation error messages to the user.
    3. Present system error messages to the user.

Present Fee and Options

If the Get Fee Configuration step determines that there is a fee payable at submission time then this page flow must be presented to the applicant. Fees are calculated by Student Financials according to the Fee Code attached to the Application Center. This page flow needs to present the applicant with his/her payment options based on the values returned in the earlier Get Fee Configuration step. The options here will include the ability to specify the tender category (CC, check etc) and optionally the ability to request a Waiver if that is allowed. Partial payments are not allowed. Therefore the self service online application must not allow the applicant to change amount presented for payment.

A typical self service online application UI might perform the following actions:

- Present a page showing the:
  1. Amount payable.
  2. Currency code of the fee amount.
3. Optionally the ability to request a Waiver.
4. Optionally the ability to specify a Waiver Basis code to be associated with the Waiver request.
5. Optionally the ability for the applicant to attach one or more attachments to be used as supporting documentation for the Waiver request.

- Formulate a request message based on the choice of the applicant:
  1. If the applicant requests a Waiver then:
     - Add Waiver details to the request
     - Submit the request to the SAD_SUBMITAPPL Web Service
     - Process the response as in the Submit Application flow
  2. If the applicant chooses to proceed with payment
     - Submit the request message to the SSF_INITIATE_EPAYMENT web service
     - Receive and process the response from the payment service
     - The response may be either a HTTP redirect OR a HTML string (this is determined by the hosted payment configuration)
     - In the case of a redirect the UI needs to respect the HTTP redirect and it will also need to store state of the current logged on userid/password in persistent client storage (e.g. a cookie or database table)
     - In the case of a HTML string response the UI can handle this in a number of ways, some examples being:
       - Spawn a new child window to display the HTML string response
       - Incorporate the HTML response into a parent window frame
  3. The response style is configurable and customizable and by use of dynamic HTML. There is sufficient flexibility for the UI to initiate the Payment Processing flow in either a new or an existing browser window. The decision will hinge entirely on the desired user experience and the technical capabilities of the UI technology being employed.

- The end result of this activity is that the user will be presented with a browser page provided by the 3rd party Hosted Payment processor from where the applicant can proceed with payment as given in the Payment Processing flow.

**Payment Processing**

At this point the applicant has indicated that they wish to proceed with payment of a fee and they will have been presented with a browser window which contains a HTML page provided by the 3rd party hosted payment provider (e.g. the applicant will have been redirected to the 3rd party site).

The applicant would then proceed to enter payment options and submit or cancel the payment. The 3rd party provider would then redirect the applicant’s browser to one of three configured URLs:

- CANCEL – the URL destination for when the applicant cancels the payment
- SUCCESS – the URL destination for when the payment is completed successfully
- FAIL - the URL destination for when the payment is not completed successfully

In the event of a successful payment then one final step needs to occur in order to complete the payment cycle. The Complete ePayment web service needs to be invoked to finalize the payment and submit the application. The self service online application UI should display the appropriate message to the applicant based on the Complete ePayment (SSF_COMPLETE_EPAYMENT) response message.

A typical self service online application UI might perform the following actions:

- Formulate a request message
- Submit the request to the SSF_COMPLETE_EPAYMENT web service
• Present a page to notify the applicant of successful payment
• Optionally transfer the applicant back to the Online Application home page and initiate the Get Application Summary flow to show the latest status.

ERROR HANDLING

When the system validates the data during save and submit, if validation errors occur, the error handling feature returns a list of textual descriptions of those errors along with error IDs and the field names related to the errors. As much information as possible is returned to the user interface to present the errors to the user, and allow correction of the errors.

The user interface must perform the following steps to use the error handling feature:

• The UI must populate a unique ID into the tag called SCC_ENTITY_INST_ID in the request xml for each entity. In the event of error during data validation, error handling returns to the UI the unique ID (that is obtained from the request message) and the field name related to the error along with the error message text.
• The UI can use this unique error ID, the field name, and the error message to know what fields on what pages are in error. This allows the UI to highlight the fields with errors and allows the user to correct the data.

The unique ID is optional in the request schema. Therefore, institutions should populate the unique ID only if they want to uptake the error handling feature.

Institutions are free to populate the SCC_ENTITY_INST_ID in any format they please, but it is recommended that they use the official UUID format, 36 character ISO standard found at http://en.wikipedia.org/wiki/Universally_unique_identifier

LOGOFF

An option to leave the online application could potentially be provided from any page in the online application. Clicking the Logout hyperlink is a signal from the users that they intend to leave the system and that any sensitive information managed by the user interface should now be cleared.

A typical self service online application UI might perform the following actions:

• Present a "Logout" hyperlink that the user can click to log out of the online application.
• Clear the username and password combination from the local UI temporary storage area.
• Present the user with a logged off confirmation page.

Building the Hosted Payment HTML object

For a detailed account of the Hosted payment feature please see the Electronic Payment Integration Developers Reference Guide found at the following location:


The HTML object is provided as a flexible means to allow a diverse range of UI technologies to interact with the Hosted payment feature. The concept is that the HTML object can be constructed using dynamic HTML. The HTML object can then be presented in a browser window to allow seamless flow between the Admissions Online Application and the 3rd party payment provider. In theory there should
be no page flow or User Experience which cannot be achieved using a JavaScript based HTML object. Note that the HTML object is not limited to JavaScript and can contain any proprietary client side scripting language as is dictated by the UI technology in use by the institution.

As a simple example the following HTML object could be used to auto-redirect the applicant’s browser to a 3rd party payment provider:

```html
<BODY onload="submitForm()">
<form name="myform" action="https://someserver/upay/web/index.jsp" method="post">
<input type="hidden" name="UPAY_SITE_ID" value="3">
<input type="hidden" name="TICKET" value="12345">
<input type="hidden" name="TICKET_NAME" value="12345">
</form>
<script LANGUAGE="JavaScript">function submitForm()

{document.myform.submit();}
</script>
</BODY>
```
Avoiding caching issues:

Your web server or browser may cache old content. One way to overcome this issue is to enclose the online application page content in a HTML wrapper as shown in the following code:

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
<!-- BEGIN INSERT --><META HTTP-EQUIV="Expires" CONTENT="Mon, 04 Dec 1999 21:29:02 GMT"><!-- END INSERT -->
<!-- BEGIN INSERT --><HEAD><META HTTP-EQUIV="PRAGMA" CONTENT="NO-CACHE"></HEAD><!--END INSERT -->
<title>OLA Demo</title>
</head>
<body bgcolor=white lang=EN-US link="#3366cc" vlink="#9999cc" style='tab-interval: .5in' leftmargin=10 RIGHTMARGIN=10 alink="#0000cc">
<object width="800" height="500">
<param name="movie" value="OLA.swf">
<embed src="OLA.swf" width="800" height="500">
</embed>
</object>
</body>
</html>
```

Microsoft Internet Explorer 7 Trusted sites Issue:

When using Microsoft Internet Explorer 7, you may need to add your web server URL hostname to Internet Explorer 7's trusted sites. If you do not add the URL to the trusted sites, you will receive a security error: "Error in line 1 char 2 .......". This is a rather cryptic error, but adding your web server host to the trusted sites setup in your browser will correct this issue.

Adobe Flex Domain Security Issue:

This issue is applicable if you are using Adobe Flex to develop an online application.

If the user loads this online application from a particular domain, the online application cannot retrieve data from another domain. To avoid this issue, you can create a crossdomain.xml file that allows your online application to load content from other domains. Refer to the Adobe Flash or Adobe Flex website for more information on the crossdomain.xml file.
APPENDIX: LOADING APPLICATIONS IN BULK

This chapter presents a scenario where you use the File Parser, AAWS, and Transaction Manager features to load applications into Campus Solutions from a text file. In this scenario we are using a subset of admissions application data found in a typical application load file.

To load applications from an external file:

1. Set up File Parser definitions.
2. Set up the Constituent transaction.
3. Run the File Parser process to load the data into staging tables.
4. Review the staged data.
5. Run the Search Match Post process.

Setting Up File Parser Definitions

Steps:

- Create the File Parser definitions.
- Make any necessary additions and/or deletions to the File Parser definitions.

Step 1: Create the File Parser definitions.

The File Parser definition presented in the following sections serves as an example of how an institution might configure a bulk load.

Step 2: Make any necessary changes to the File Parser definitions:

Note: For more information on File Parser, see PeopleSoft Enterprise Campus Community Fundamentals 9.0 People Book: "Using the File Parser Process".

The File Parser definitions consist of several key components:
• **Field Conversion Definitions** – Identifies external file field values to convert before storing the data in the target staging table.

• **Context Definitions** – Identifies the parent-to-child relationship of the staging tables that will be used to hold the data.

• **File Mapping Definitions** – Associates fields on the external file with the context definition.

Use the Field Conversion Definition page to define field values in the external file that must be converted to internal values. Add any additional fields and complete the definition for the fields delivered. To navigate to the Field Conversion Definition page, select *Set Up SACR, System Administration, Utilities, File Parser, Field Conversion Definition*.

![Field Conversion Definition page](image.png)

**Figure 49: Field Conversion Definition page**

Use the Context Definition page to add any additional extension staging tables that need to be updated when loading data from the external file. You may have created these tables by performing steps listed in the Extending AAWS chapter. Ensure that you enter the correct Parent Record field value. To navigate to the Context Definition page, select *Set Up SACR, System Administration, Utilities, File Parser, Context Definition*.
Use the File Definition page to provide information about the external file. To navigate to the File Definition page, select Set Up SACR, System Administration, Utilities, File Parser, File Mapping Definition.

Use the File Layout page to define fields from the external file that will be loaded into the staging tables.
Figure 52: File Layout page

Use the Mapping page to associate fields defined on the File Layout page with staging table fields defined in the context definition.

Figure 53: Mapping page

Use the Preview Data page to test your external file with the File Parser Definitions. **Note**: The preview feature shows only the first row of each row type; therefore, you should use a small test file.
Setting up Constituent Transaction

After you set up the file parser, set up a Transaction Manager Constituent transaction that represents data in the external file.

Use the Transaction Setup page to configure the Stage and Load transaction. To navigate to the Transaction Setup page, select Set Up SACR, System Administration, Utilities, Constituent Transaction Mgmt, Transaction Setup.
Enter the Implementation Class Details as shown in the above screenshot.

Ensure that the following parameters are set up:

- Transaction Status is set to **Active**.
- The Process Search/Match and the Batch options are selected.

### Running the File Parser Process

Run the File Parser process to load the application data from a text file into the staging tables.

In order to run the File Parser process, set up a run control with the File Mapping ID and filename information on the Run File Parser page. To navigate to the Run File Parser page, select *Set Up SACR, System Administration, Utilities, File Parser, Run File Parser.*
Reviewing Staged Data

After loading the application data into the staging tables, perform the following steps:

Steps:

- Review the staged constituent data for the ADM_BATCH_APPLICANTS transaction.
- Review the staged application data.

**Step 1**: Review the staged constituent data:

To navigate to the Constituent Staging component, select *Student Admissions, Application Transaction Mgmt, Constituent Staging*.

Enter the ADM_BATCH_APPLICANTS transaction code in the Constituent Staging search component to review the loaded data:
Use the Constituent Staging component to review and correct the constituent data:

**Figure 57: Constituent Staging component**

Save the data after review.

**Step 2:** Review the Staged Application Data:

After reviewing the constituent data, use the Application Transactions component to review and correct the application data.

To navigate to the Application Transactions component, click the View Transaction Data link in the Constituent Staging component or, alternatively, select *Student Admissions, Application Transaction Mgmt, Application Transactions*.

Save any changes.
Running the Search Match Post Process

After loading application data into staging tables, reviewing and correcting the staging data, the data needs to be posted into the Campus Solutions production tables.


Figure 58: Application Transactions component

Figure 59: Transaction Management Process component
It is important to note the following:

- In the screen shot, search match post process is performed only for Submitted data, which was loaded in the previous step.

- The process will run Search Match and, based on a match, post the person and application transaction data.
Application Class Reference

Entity Interface – IEntity

SCC_COMMON:ENTITY

Interface IEntity Implementing Classes:

FileAttachment, AbstractEntity

IEntity is the interface that all entities must implement at some level. While it can be implemented directly, it is strongly recommended to make use of the base classes BasicEntity, StagedEntity or StagedHREntity.

<table>
<thead>
<tr>
<th>Property Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public Array&lt;ChildEntity&gt; ChildEntityArray</strong></td>
</tr>
<tr>
<td><strong>public Record data</strong></td>
</tr>
<tr>
<td><strong>public boolean entityDelete</strong></td>
</tr>
<tr>
<td><strong>public string entityId</strong></td>
</tr>
<tr>
<td><strong>public string entityName</strong></td>
</tr>
<tr>
<td><strong>public IEntity parent</strong></td>
</tr>
<tr>
<td><strong>public ChildEntity parentCE</strong></td>
</tr>
<tr>
<td><strong>public boolean selfServiceMode</strong></td>
</tr>
<tr>
<td><strong>public boolean STAGE_MODE</strong></td>
</tr>
<tr>
<td>public string UPDATE_RULE</td>
</tr>
<tr>
<td>---------------------------</td>
</tr>
<tr>
<td>Specifies the update rule to use for the purposes of Data Update Rule</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public string USER_CONTEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>The user that this is being run on behalf of, since adds and updates may be done from administrative mode on behalf of a given user</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public Array of string XSDFieldList</th>
</tr>
</thead>
<tbody>
<tr>
<td>A string list of all the fields for the current entity for the XSD. This is used in conjunction with buildXSD to determine if key fields are required, based on if their parent has the same key</td>
</tr>
</tbody>
</table>

## Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void buildXSD (XmlNode xmlnode)</td>
<td>Builds the xml schema section pertaining to this entity</td>
</tr>
<tr>
<td>public void delete (boolean p_entityDelete)</td>
<td>Sets entityDelete property for this entity and all its children</td>
</tr>
<tr>
<td>public void demote (number p_tempID)</td>
<td>Moves data from production records to staging record</td>
</tr>
<tr>
<td>public void fill (Record p_inRec out, boolean p_clearState)</td>
<td>Fill (Populate) this entity using the given record, and populates all children based on the key structure</td>
</tr>
<tr>
<td>public void fromXmlNode (XmlNode parentNode)</td>
<td>Converts data from XML to the entity model</td>
</tr>
<tr>
<td>public Rowset generateRowset ( )</td>
<td>This method generates a rowset of data for the underlying record of this entity based on the keys of the parent. It is used as part of the fill process for building out the entity tree. The method is called as if it were a static method, so it is not associated with any particular entity data. The base implementation dynamically matches keys from the parent to the child and builds the query for the rowset based on that.</td>
</tr>
<tr>
<td>public string generateXSD ( )</td>
<td>Begins generation of an xml schema, based on the registry using this entity as the root</td>
</tr>
<tr>
<td>public ChildEntity getChildEntity (string p_entityID)</td>
<td>Retrieves the ChildEntity Object based on the entity ID</td>
</tr>
<tr>
<td>public number getMaxSeqNbrSQL ( )</td>
<td>Retrieves the maximum sequence number from the database based on the keystructure of the current entity</td>
</tr>
<tr>
<td>public void hardDelete ( MessageLog p_messageLog out)</td>
<td>Performs the actual delete of all objects marked with the entityDelete flag</td>
</tr>
<tr>
<td>public boolean hasEffdt ( )</td>
<td>Checks if this entity has an effective date</td>
</tr>
<tr>
<td>public boolean hasEffseq ( )</td>
<td>Checks if this entity has an effective sequence</td>
</tr>
<tr>
<td>public boolean hasKey (string p_keyname)</td>
<td>Checks if the current entity has a particular field as a key</td>
</tr>
</tbody>
</table>
public boolean isEqualKeys ( IEntity p_entity)
Determines Entity Key Equality

public void prePublish (Rowset p_changesRS out)
Uses the workflowPreimage and post images of the entity to build a PSCAMA Rowset in preparation for publish. The parameter Rowset may be Null. Data should be appended to this rowset if it is NON-Null.

public void preSave ( )
Performs pre-Save activities on this entity

public void preSaveChildren ( )

public void preSaveFirst ( )
Pre-Save code that only runs on the first entity of a child array

public void promote (string emplID, MessageLog p_log out)
Moves Data from Staging to Production Records (does not validate or save)

public void publish (Rowset p_changesRS, boolean p_onlineMode)
Takes a PSCAMA based rowset of information to be published during workflow Uses that information to construct a Message and according to the onlineMode publishes the Message data in realtime or to the batch delay table

public void save ( MessageLog p_messageLog out)
Saves this entity and all its children

public void setChildren ( )
Builds out the structure of the ChildEntityArray object based on the registry

public void setDefault ( )
Sets default values for record fields

public void setStageMode (boolean p_stageInd)
Changes the Stage_Mode property, as well as any related records. This should not be used once data has been set; to convert data, use promote or demote.

public void setUserContext (string p_userContext)
Sets the value of the UserContext property

public void toXmlNode (XmlNode parentNode)
Converts the data from the entity structure to XML

public void updateKeyFields ( )
Updates the keyfields of all children based on the current entity. This allows key changes to roll down properly.

public void validate ( MessageLog p_messageLog out)
Validate this entities contents (and all its children)

public void validateFirst ( MessageLog p_messageLog out)
Validation code that only runs on the first entity of a child array

public void workflow (boolean p_onlineMode)
Orchestrates the prepublish and publish methods for this entity and all children.

Property Detail

ChildEntityArray

public Array of ChildEntity
An array of all the ChildEntities of the current entity

data

readonly public Record
    This entities backing record

entityDelete

public boolean
    Marks this entity for deletion, and prevents save

entityID

readonly public string
    ID of the entity from the registry

entityName

readonly public string
    The name of the entity as defined in the registry

parent

public IEntity
    Parent Entity, null if it is at the top level of tree

parentCE

public ChildEntity
    The child entity that encompasses all the entities of a particular type under the parent

selfServiceMode

public boolean
    Self Service mode flag

STAGE_MODE

public boolean
    Specifies if this entity is currently in Stage Mode (True) and working with the stage record, or in Production Mode (False) and working with the production record
UPDATE_RULE

public string
Specifies the update rule to use for the purposes of Data Update Rule

USER_CONTEXT

public string
The user that this is being run on behalf of, since adds and updates may be done from
administrative mode on behalf of a given user

XSDFieldList

public Array of string
A string list of all the fields for the current entity for the XSD. This is used in
conjunction with buildXSD to determine if key fields are required, based on if their
parent has the same key

Method Detail

buildXSD

public void buildXSD (XmlNode xmlnode)
Builds the xml schema section pertaining to this entity

Parameters:
xmlnode - The Tools XmlNode object of the parent that should be continued
by this entity

delete

public void delete (boolean p_entityDelete)
Set entityDelete property for this entity and all its children

Parameters:
p_entityDelete - The value to be set for the entityDelete property
demote

public void demote (number p_tempID)
Moves data from production records to staging record

Parameters:
p_tempID - The tempid to replace the emplid with
fill

public void fill (Record p_inRec out, boolean p_clearState)
Fill (Populate) this entity using the given record, and populates all children based on
the key structure

Parameters:
p_inRec - This record populates the record underlying this entity
p_clearState - If the clearstate is true, the datapreimage will be populated as
well

fromXmlNode

public void fromXmlNode (XmlNode parentNode)
Converts data from XML to the entity model

Parameters:
parentNode - The XML node to pull from

generateRowset

public Rowset generateRowset ( )
This method generates a rowset of data for the underlying record of this entity based
on the keys of the parent.
It is used as part of the fill process for building out the entity tree.
The method is called as if it were a static method, so it is not associated with any
particular entity data.
The base implementation dynamically matches keys from the parent to the child and
builds the query for the rowset based on that.

Returns:
Rowset - A rowset populated based on the parent keys, or null if no rows are
found

generateXSD

public string generateXSD ( )
Begins generation of an xml schema, based on the registry using this entity as the root

Returns:
string - The XSD as a string

getChildEntity

public ChildEntity getChildEntity (string p_entityID)
Retrieves the ChildEntity Object based on the entity ID

**Parameters:**
- `p_entityID` - the entity ID, based on the registry, of the ChildEntity object to retrieve

**Returns:**
- ChildEntity

---

### getMaxSeqNbrSQL

**public number getMaxSeqNbrSQL ( )**

Retrieves the maximum sequence number from the database based on the keystructure of the current entity

**Returns:**
- number - The maximum sequence number found.

---

### hardDelete

**public void hardDelete (MessageLog p_messageLog out)**

Performs the actual delete of all objects marked with the entityDelete flag

**Parameters:**
- `p_messageLog` - The MessageLog object to log all errors/warnings to

---

### hasEffdt

**public boolean hasEffdt ( )**

Checks if this entity has an effective date

**Returns:**
- boolean - True if it has an effective date; false otherwise

---

### hasEffseq

**public boolean hasEffseq ( )**

Checks if this entity has an effective sequence

**Returns:**
- boolean - True if it has an effective sequence; false otherwise

---

### hasKey

**public boolean hasKey (string p_keyname)**

Checks if the current entity has a particular field as a key

**Parameters:**
- `p_keyname` - the key name of the field to check
p_keyname - The name of the field to check

Returns:
boolean - True if the field is a key, false if the field is not a key or the entity does not include it

### isEqualKeys

```java
def isEqualKeys(IEntity p_entity):
    # Determine Entity Key Equality
```

**Parameters:**
- `p_entity`: The entity to compare the current entity to

**Returns:**
- boolean: True if the entity keys are equal, false otherwise

### prePublish

```java
def prePublish(Rowset p_changesRS):
    # Uses the workflowPreimage and post images of the entity to build a PSCAMA Rowset in preparation for publish. The parameter Rowset may be Null. Data should be appended to this rowset if it is NON-Null.
```

**Parameters:**
- `p_changesRS`: 

### preSave

```java
def preSave():
    # Perform pre-Save activities on this entity
```

### preSaveChildren

```java
def preSaveChildren():
    # PreSave code that only runs on the first entity of a child array
```

### PreSaveFirst

```java
def PreSaveFirst():
    # Pre-Save code that only runs on the first entity of a child array
```

### promote

```java
def promote(emplID, MessageLog p_log):
    # Moves data from staging to production records (does not validate or save)
```

**Parameters:**
- `emplID`: The emplid for production to replace the tempid from staging
**publish**

```java
public void publish (Rowset p_changesRS, boolean p_onlineMode)
```

Takes a PSCAMA based rowset of information to be published during workflow; uses that information to construct a Message and, according to the onlineMode, publishes the Message data in realtime or to the batch delay table.

**Parameters:**
- `p_changesRS`
- `p_onlineMode`

**save**

```java
public void save (MessageLog p_messageLog out)
```

Save this entity and all its children.

**Parameters:**
- `p_messageLog` - The messagelog object to log errors and warnings to

**setChildren**

```java
public void setChildren ( )
```

Builds out the structure of the ChildEntityArray object based on the registry.

**setDefault**

```java
public void setDefault ( )
```

Sets default values for record fields.

**setStageMode**

```java
public void setStageMode (boolean p_stageInd)
```

Changes the Stage_Mode property, as well as any related records. This should not be used once data has been set, to convert data use promote or demote.

**Parameters:**
- `p_stageInd` - The value to set the stage mode to, if the value passed matches the current stage mode no changes are made

**setUserContext**

```java
public void setUserContext (string p_userContext)
```
Sets the value of the UserContext property

Parameters:
  p_userContext - The value set for the property UserContext

toXmlNode
d
  public void toXmlNode (XmlNode parentNode)
  Converts the data from the entity structure to XML

  Parameters:
  parentNode - The XML parent node to push the xml to

updateKeyFields
d
  public void updateKeyFields ( )
  Updates the keyfields of all children based on the current entity. This allows key
  changes to roll down properly.

validate
d
  public void validate ( MessageLog p_messageLog out)
  Validate this entities contents (and all its children)

  Parameters:
  p_messageLog - The message log object to log errors and warnings to

validateFirst
d
  public void validateFirst ( MessageLog p_messageLog out)
  Validation code that only runs on the first entity of a child array

  Parameters:
  p_messageLog - The MessageLog object to log all errors/warnings to

workflow
d
  public void workflow (boolean p_onlineMode)
  Orchestrates the prepublish and publish methods for this entity and all children.

  Parameters:
  p_onlineMode
Abstract Entity – AbstractEntity

SCC_COMMON:ENTITY
Abstract Class AbstractEntity Implemented Interfaces:
  IEntity

Direct Known Subclasses:
  StagedEntity, StagedHREntity, BasicEntity

This is the base implementation of IEntity, as it has a lot of the methods pre-built to allow for introspection based on the underlying record. This means it is adaptable; however, if the code has to be customized, it allows for that as well. It is still recommended that the user use the classes that implement this abstract, including BasicEntity, StagedEntity and StagedHREntity.

<table>
<thead>
<tr>
<th>Property Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public Array of ChildEntity</strong> ChildEntityArray</td>
</tr>
<tr>
<td>An array of all the ChildEntities of the current entity. The childEntities are determined by the entity registry, and populated when the entity is instantiated.</td>
</tr>
<tr>
<td><strong>public Record data</strong></td>
</tr>
<tr>
<td>The backing record for this entity.</td>
</tr>
<tr>
<td><strong>public Record dataPreimage</strong> readonly</td>
</tr>
<tr>
<td>The preimage of the backing record for this entity. The entity is compare against the preimage prior to save.</td>
</tr>
<tr>
<td><strong>public boolean DO_SAVE</strong></td>
</tr>
<tr>
<td>Specifies if the entity is allowed to save</td>
</tr>
<tr>
<td><strong>public boolean entityDelete</strong></td>
</tr>
<tr>
<td>Marks this entity for deletion, and prevents save</td>
</tr>
<tr>
<td><strong>public string entityId</strong> readonly</td>
</tr>
<tr>
<td>ID of the entity from the registry</td>
</tr>
<tr>
<td><strong>public string entityName</strong> readonly</td>
</tr>
<tr>
<td>The name of the entity as defined in the registry</td>
</tr>
<tr>
<td><strong>public ParentEntity parent</strong></td>
</tr>
<tr>
<td>The parent Entity, null if it is at the top level of tree</td>
</tr>
<tr>
<td><strong>public ChildEntity parentCE</strong></td>
</tr>
<tr>
<td>The child entity that encompasses all the entities of a particular type under the parent</td>
</tr>
<tr>
<td><strong>public string prodRecordName</strong> readonly</td>
</tr>
<tr>
<td>The name of the production record is populated here based on the registry</td>
</tr>
<tr>
<td><strong>public SCC_ROW_ADD_DTTM</strong></td>
</tr>
<tr>
<td>The Datetime the row of data was added, this is part of the who columns. This property corresponds to this field, if it exists on the record; if not, it does nothing. RECOMMENDATION: Do not override get &amp; set.</td>
</tr>
<tr>
<td><strong>public SCC_ROW_ADD_OPRID</strong></td>
</tr>
<tr>
<td>The Person who added the row of data, this is part of the who columns. This property corresponds to this field, if it exists on the record; if not, it does nothing.</td>
</tr>
</tbody>
</table>
RECOMMENDATION: Do not override get & set.

public datetime SCC_ROW_UPD_DTTM
The Datetime the row of data was updated. This is part of the who columns. This property corresponds to this field, if it exists on the record; if not, it does nothing. RECOMMENDATION: Do not override get & set.

public string SCC_ROW_UPD_OPRID
The Person who updated the row of data. This is part of the who columns. This property corresponds to this field, if it exists on the record; if not, it does nothing. RECOMMENDATION: Do not override get & set.

public boolean selfServiceMode
If the entity is in selfServiceMode

public boolean STAGE_MODE
Specifies if this entity is currently in Stage Mode (True) and working with the stage record, or in Production Mode (False) and working with the production record.

public string UPDATE_RULE
Specifies the update rule to use for the purposes of Data Update Rule.

public string USER_CONTEXT
The user that this is being run on behalf of, since adds and updates may be done from administrative mode on behalf of a given user, or %UserId.

public workflowPreimage readonly
The pre-image object to use for workflow.

public Array of string XSDFieldList
A string list of all the fields for the current entity for the XSD. This is used in conjunction with buildXSD to determine if key fields are required, based on if their parent has the same key.

protected boolean hasAudit
A value, set by testForAudit, that specifies if this entity has audit fields. It is used during save time to determine if the audit fields need to be updated.

protected Array of string ignoreFields
The array of names of fields to ignore for the purposes of moving data to/from xml, pushing a new field onto this array will add it to the list.

protected string RECORD_NAME
The name of the current record underlying this entity.

protected string XSDNS readonly
The namespace to use for the schema, found and set during generateXSD.

Constructor Summary

public AbstractEntity ( IEntity p_parent)
The Constructor.

Method Summary

public void buildXSD (XmlNode xmNode)
Builds the xml schema section pertaining to this entity.
The implementation builds and populates the schema based on the underlying record, ignoring any fields in the ignore array.
Keys that roll down from the parent will not be marked required on the child.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public void delete(boolean p_entityDelete)</code></td>
<td>Marks this entity and all its children for deletion at save time - soft delete. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><code>abstract public void demote(number p_tempID)</code></td>
<td>Moves data from production records to staging record. RECOMMENDATION: This must be implemented, but is only used if the entity is staged. In the implementation make sure to call demoteChildren to cause demote to work with the registry.</td>
</tr>
<tr>
<td><code>public boolean entityFieldUpdateNeeded(Rowset p_DataUpdateRule, string p_fieldName, any oldValue)</code></td>
<td>RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><code>public boolean entityTypeUpdateNeeded(Rowset p_DataUpdateRule, any p_entityType, boolean p_newFlag)</code></td>
<td>RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><code>public boolean entityTypeUpdateRequested(Rowset p_DataUpdateRule, any p_entityType)</code></td>
<td>RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><code>public boolean entityUpdateRequested(Rowset p_DataUpdateRule)</code></td>
<td>RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><code>public void fill(Record p_inRec out, boolean p_clearState)</code></td>
<td>Populates (fills) this entity from the given record. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><code>public void fromXmlNode(XmlNode parentNode)</code></td>
<td>Retrieves data to properties based on the fields in the record. Fields marked ignore are not Processed. Entity Element Names, wrapping of the ChildEntities or embedding is based on the entity registry. RECOMMENDATION: Only override if the xml for this entity has to be custom.</td>
</tr>
<tr>
<td><code>public Rowset generateRowset()</code></td>
<td>This method generates a rowset of data for the underlying record of this entity based on the keys of the parent. It is used as part of the fill process for building out the entity tree. The method is called as if it were a static method, so it is not associated with any particular entity data. The base implementation dynamically matches keys from the parent to the child and builds the query for the rowset based on that. RECOMMENDATION: If the key structure between the parent and child does not match up, or the rowset needs to be built using another method (i.e. a service), override this method.</td>
</tr>
<tr>
<td><code>public string generateXSD()</code></td>
<td>Generates the xml schema with this node as the root. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><code>public getChildEntity(string p_entityID)</code></td>
<td></td>
</tr>
<tr>
<td>ChildEntity</td>
<td>Retrieves the ChildEntity Object based on the entity ID</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>public number</td>
<td>getMaxSeqNbrSQL ( )</td>
</tr>
<tr>
<td>public void</td>
<td>hardDelete ( MessageLog p_messageLog out)</td>
</tr>
<tr>
<td>public boolean</td>
<td>hasEffdt ( )</td>
</tr>
<tr>
<td>public boolean</td>
<td>hasEffseq ( )</td>
</tr>
<tr>
<td>public boolean</td>
<td>hasKey (string p_keyname)</td>
</tr>
<tr>
<td>public boolean</td>
<td>isEqualKeys ( IEntity p_entity)</td>
</tr>
<tr>
<td>public void</td>
<td>populateAudit ( )</td>
</tr>
<tr>
<td>public void</td>
<td>prePublish (Rowset p_changesRS out)</td>
</tr>
<tr>
<td>public void</td>
<td>preSave ( )</td>
</tr>
<tr>
<td>public void</td>
<td>preSaveChildren ( )</td>
</tr>
<tr>
<td>public void</td>
<td>preSaveFirst ( )</td>
</tr>
</tbody>
</table>
| abstract public void | promote (string p_emplID, MessageLog p_log out) | Moves Data from Staging to Production Records (does not validate or save) RECOMMENDATION: This must be implemented, but is only used if the entity is staged. In the
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public void</strong> publish Rowset p_changesRS, boolean p_onlineMode**</td>
<td>Takes a PSCAMA based rowset of information to be published during workflow; uses that information to construct a Message and, according to the onlineMode, publishes the Message data in realtime or to the batch delay table. RECOMMENDATION: Override if publish logic is required.</td>
</tr>
<tr>
<td><strong>public Rowset</strong> retrieveDataRule()</td>
<td>Uses the entityID and Update Rule Name to read Data Update Rule information. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><strong>public void</strong> save MessageLog p_messageLog out</td>
<td>Base implementation, calls presave and issues save for all children, then a save for this entity. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><strong>public void</strong> setChildren()</td>
<td>Builds out the ChildEntityArray based on the entity registry. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><strong>public void</strong> setDefault()</td>
<td>Base implementation sets the default value for each field using the Field setDefault method if the field has not already been marked changed. RECOMMENDATION: Override if required.</td>
</tr>
<tr>
<td><strong>public void</strong> setStageMode boolean p_stageInd</td>
<td>Sets the stage mode and changes the underlying record to stage/production for this entity and all children. This implementation does nothing since stage mode is not required. RECOMMENDATION: Override if the entity is Staged.</td>
</tr>
<tr>
<td><strong>public void</strong> setUpdateRule string p_UpdateRule</td>
<td>Sets a specific update rule. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><strong>public void</strong> setUpdateRuleByTransName string p_transName</td>
<td>Sets the update rule based on the transaction name in the setup table. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><strong>public void</strong> setUserContext string p_userContext</td>
<td>Sets the user context for this entity. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><strong>public void</strong> toXmlNode XmlNode parentNode</td>
<td>Pushes data to the XML structure from the entity. Fields marked ignore are not Processed. Entity Element Names, wrapping of the ChildEntities or embedding is based on the entity registry. The value of properties corresponding to the fields are pushed. RECOMMENDATION: Only override if the xml for this entity has to be custom.</td>
</tr>
<tr>
<td><strong>public void</strong> updateChildFields string propname, array of ChildEntity cea</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Sets the value of a specific property on all ChildEntities to match the current entity</strong> RECOMMENDATION: Do not override.</td>
<td></td>
</tr>
<tr>
<td><strong>public void validate (MessageLog p_messageLog out)</strong></td>
<td>Validates the entity object if entity is not marked for deletion RECOMMENDATION: Override this method if the entity requires custom validation. Use commonValidate to call the system edits. This method does the following standard system edits: - Required fields are present - Validates all 1/0 fields contain only a 1 or a 0 - Validates all translate fields have a valid value - Validates all YesNo fields contain a Y or an N - Validates all prompt edit fields have a valid value</td>
</tr>
<tr>
<td><strong>public void validateFirst (MessageLog p_messageLog out)</strong></td>
<td>Allows validations to be performed on all entities under a childEntity. This is valuable in doing validation against other entities under the same childEntity. RECOMMENDATION: Override this method if the entity requires custom validation crossing entity boundaries.</td>
</tr>
<tr>
<td><strong>public void workflow (boolean p_onlineMode)</strong></td>
<td>Orchestrates the prepublish and publish methods for this entity and all children. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><strong>protected void buildXSDChildren (XmlNode xmlnode)</strong></td>
<td>Iterates through the ChildEntityArray and issues a buildXSD against each ChildEntity, which in turn creates a temporary entity and calls buildXSD. Base implementation uses the entity registry. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><strong>protected void changeRecord (string p_recname)</strong></td>
<td>Changes the record to the record name specified. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><strong>protected void commonValidate (MessageLog p_messageLog out, number p_validateArgs)</strong></td>
<td>Runs the record based validations and outputs results to the log. Validation Arguments</td>
</tr>
<tr>
<td>%Edit_DateRange</td>
<td>Reasonable Date Range (Is the date contained within the specified reasonable date range?)</td>
</tr>
<tr>
<td>%Edit_OneZero</td>
<td>1/0 (Do all 1/0 fields contain only a 1 or 0?)</td>
</tr>
<tr>
<td>%Edit_PromptTable</td>
<td>Prompt Table (Is field data contained in the specified prompt table?)</td>
</tr>
<tr>
<td>%Edit_Required</td>
<td>Required Field (Do all required fields contain data? For numeric or signed fields, it checks that they do not contain NULL or 0 values.)</td>
</tr>
<tr>
<td>%Edit_TranslateTable</td>
<td>Translate Table (Is field data contained in the specified translate table?)</td>
</tr>
<tr>
<td>%Edit_YesNo</td>
<td>Yes/No (Do all yes/no fields contain only yes or no data?)</td>
</tr>
<tr>
<td><strong>protected void defaultChildren ( )</strong></td>
<td>Iterates through the ChildEntityArray and issues a setDefault against each ChildEntity, which in turn calls setDefault on all of the entities it contains. Base implementation uses the entity registry. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td><strong>protected void deleteChildren (boolean p_entityDelete)</strong></td>
<td></td>
</tr>
<tr>
<td>Method Details</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>Iterates through the ChildEntityArray and issues a delete against each ChildEntity, which in turn calls delete on all of the entities it contains. Base implementation uses the entity registry. RECOMMENDATION: Do not override.</td>
<td></td>
</tr>
<tr>
<td>protected void <code>demoateChildren</code> (number <code>p_tempID</code>)</td>
<td></td>
</tr>
<tr>
<td>Iterates through the ChildEntityArray and issues a demote against each ChildEntity, which in turn calls demote on all of the entities it contains. Base implementation uses the entity registry. RECOMMENDATION: Do not override.</td>
<td></td>
</tr>
<tr>
<td>protected void <code>fillChildren</code> (boolean <code>p_clearState</code>)</td>
<td></td>
</tr>
<tr>
<td>Iterates through the ChildEntityArray and issues a fill against each ChildEntity, which in turn calls fill on all of the entities it contains. Base implementation uses the entity registry. RECOMMENDATION: Do not override.</td>
<td></td>
</tr>
<tr>
<td>protected void <code>fromXmlNodeChildren</code> (XmlNode <code>parentNode</code>)</td>
<td></td>
</tr>
<tr>
<td>Iterates through the ChildEntityArray and issues a fromXMLNode against each ChildEntity, which in turn calls fromXMLNode on all of the entities it contains. Base implementation uses the entity registry. RECOMMENDATION: Do not override.</td>
<td></td>
</tr>
<tr>
<td>protected <code>getCSUserDefaults</code> ()</td>
<td></td>
</tr>
<tr>
<td>Retrieves the CSUserDefaults RECOMMENDATION: Do not override.</td>
<td></td>
</tr>
<tr>
<td>protected <code>getInstallationHCM</code> ()</td>
<td></td>
</tr>
<tr>
<td>Retrieves the HCM Installation Defaults RECOMMENDATION: Do not override.</td>
<td></td>
</tr>
<tr>
<td>protected <code>getParent</code> ()</td>
<td></td>
</tr>
<tr>
<td>Retrieves the parent Entity RECOMMENDATION: Do not override.</td>
<td></td>
</tr>
<tr>
<td>protected <code>getRecordName</code> ()</td>
<td></td>
</tr>
<tr>
<td>Retrieves the name of the current record on this entity. RECOMMENDATION: Do not override.</td>
<td></td>
</tr>
<tr>
<td>protected void <code>hardDeleteChildren</code> (MessageLog <code>p_messageLog</code> out)</td>
<td></td>
</tr>
<tr>
<td>Iterates through the ChildEntityArray and issues a hardDelete against each ChildEntity, which in turn calls hardDelete on all of the entities it contains. Base implementation uses the entity registry. RECOMMENDATION: Do not override.</td>
<td></td>
</tr>
<tr>
<td>protected void <code>promoteChildren</code> (string <code>p_emplID</code>, MessageLog <code>p_log</code> out)</td>
<td></td>
</tr>
<tr>
<td>Iterates through the ChildEntityArray and issues a promote against each ChildEntity, which in turn calls promote on all of the entities it contains. Base implementation uses the entity registry. RECOMMENDATION: Do not override.</td>
<td></td>
</tr>
<tr>
<td>protected void <code>saveChildren</code> (MessageLog <code>p_messageLog</code> out)</td>
<td></td>
</tr>
</tbody>
</table>
Iterates through the ChildEntityArray and issues a Save against each ChildEntity, which in turn calls Save on all of the entities it contains. Base implementation uses the entity registry. RECOMMENDATION: Do not override.

protected void setData (Record p_dataRec)

Sets the Data record
RECOMMENDATION: Do not override.

protected void setDataPreImage (Record p_dataPreImageRec)

Sets the Data preimage
RECOMMENDATION: Do not override.

protected void setEntityID (string p_entityID)

Sets the entity ID, and retrieves the related information from the registry
RECOMMENDATION: Do not override.

protected void setKeyList ()

Sets the list of key fields to private instance variable keyList
RECOMMENDATION: Do not override.

protected void testForAudit ()

Checks if the underlying record has the audit fields and sets the property hasAudit.
RECOMMENDATION: Do not override.

protected void toXmlNodeChildren (XmlNode parentNode)

Iterates through the ChildEntityArray and issues a toXMLNode against each ChildEntity, which in turn calls toXMLNode on all of the entities it contains. Base implementation uses the entity registry. RECOMMENDATION: Do not override.

protected void updateKeyFields ()

Updates all the key fields of the child entities that match the key fields on this entity
RECOMMENDATION: Do not override.

protected void validateChildren (MessageLog p_messageLog out)

Iterates through the ChildEntityArray and issues a validate against each ChildEntity, which in turn calls validate on all of the entities it contains. Base implementation uses the entity registry. RECOMMENDATION: Do not override.

Property Detail

ChildEntityArray

public Array of ChildEntity

An array of all the ChildEntities of the current entity. The childEntities are determined by the entity registry, and populated when the entity is instansiated.
readonly public Record
    The backing record for this entity.

dataPreimage
readonly public Record
    The preimage of the backing record for this entity. The entity is compare against the
    preimage prior to save.

DO_SAVE
public boolean
    Specifies if the entity is allowed to save

tentityDelete
public boolean
    Marks this entity for deletion, and prevents save

tenentityID
readonly public string
    ID of the entity from the registry

tenentityName
readonly public string
    The name of the entity as defined in the registry

tenparent
    The parent Entity, null if it is at the top level of tree

tenparentCE
    The child entity that encompasses all the entities of a particular type under the parent

prodRecordName
readonly public string
    The name of the production record is populated here based on the registry
SCC_ROW_ADD_DTTM

public datetime
The Datetime the row of data was added, this is part of the who columns. This property corresponds to this field, if it exists on the record; if not, it does nothing.
RECOMMENDATION: Do not override get & set.

SCC_ROW_ADD_OPRID

public string
The Person who added the row of data, this is part of the who columns. This property corresponds to this field, if it exists on the record; if not, it does nothing.
RECOMMENDATION: Do not override get & set.

SCC_ROW_UPD_DTTM

public datetime
The Datetime the row of data was update, this is part of the who columns. This property corresponds to this field, if it exists on the record; if not, it does nothing.
RECOMMENDATION: Do not override get & set.

SCC_ROW_UPD_OPRID

public string
The Person who updated the row of data, this is part of the who columns. This property corresponds to this field, if it exists on the record; if not, it does nothing.
RECOMMENDATION: Do not override get & set.

selfServiceMode

public boolean
If the entity is in selfServiceMode

STAGE_MODE

public boolean
Specifies if this entity is currently in Stage Mode (True) and working with the stage record, or in Production Mode (False) and working with the production record

UPDATE_RULE

public string
Specifies the update rule to use for the purposes of Data Update Rule
public string
   The user that this is being run on behalf of, since adds and updates may be done from
   administrative mode on behalf of a given user, or %UserId.

workflowPreimage
readonly public Record
   The pre-image object to use for workflow

XSDFieldList
public Array of string
   A string list of all the fields for the current entity for the XSD. This is used in
   conjunction with buildXSD to determine if key fields are required, based on if their
   parent has the same key

hasAudit
protected boolean
   A value, set by testForAudit, that specifies if this entity has audit fields. It is used
   during save time to determine if the audit fields need to be updated.

ignoreFields
protected Array of string
   The array of names of fields to ignore for the purposes of moving data to/from xml,
   pushing a new field onto this array will add it to the list

RECORD_NAME
protected string
   The name of the current record underlying this entity.

XSDNS
readonly protected string
   The namespace to use for the schema, found and set during generateXSD

Constructor Detail

AbstractEntity
public AbstractEntity ( IEntity p_parent)
The Constructor

Parameters:
   p_parent - The parent of this entity, or null if this is the top of the entity tree

Method Detail

buildXSD

public void buildXSD (XmlNode xmlnode)
Builds the xml schema section pertaining to this entity.
The implementation builds and populates the schema based on the underlying record, ignoring any fields in the ignore array.
Keys that roll down from the parent will not be marked required on the child.
RECOMMENDATION: Only override if the entity requires a custom schema.

Parameters:
   xmlnode - The Tools XmlNode object of the parent that should be continued by this entity

delete

public void delete (boolean p_entityDelete)
Marks this entity and all its children for deletion at save time - soft delete.
RECOMMENDATION: Do not override.

Parameters:
   p_entityDelete - True to delete this entity and False to undelete.

demote

abstract public void demote (number p_tempID)
Moves data from production records to staging record
RECOMMENDATION: This must be implemented, but is only used if the entity is staged. In the implementation make sure to call demoteChildren to cause demote to work with the registry.

Parameters:
   p_tempID - The tempid to replace the emplid with

delteFieldUpdateNeeded

public boolean delteFieldUpdateNeeded (Rowset p_DataUpdateRule, string p_fieldName, any oldValue)

RECOMMENDATION: Do not override.
Parameters:
   p_DataUpdateRule
   p_fieldName
   oldValue

Returns:
   boolean

**entityTypeUpdateNeeded**

public boolean entityTypeUpdateNeeded (Rowset p_DataUpdateRule, any p_entityType, boolean p_newFlag)

RECOMMENDATION: Do not override.

Parameters:
   p_DataUpdateRule
   p_entityType
   p_newFlag

Returns:
   boolean - True if the Data Update Rule specifies the entity type should be updated

**entityTypeUpdateRequested**

public boolean entityTypeUpdateRequested (Rowset p_DataUpdateRule, any p_entityType)

RECOMMENDATION: Do not override.

Parameters:
   p_DataUpdateRule - The data update rule
   p_entityType - The type specifying whether this is a Typed, Non-Typed or HR entity

Returns:
   boolean - True if any row of the Data Update Rule Detail (for this entity AND type) has an action not equal to "Do Not Update" (N)

**entityUpdateRequested**

public boolean entityUpdateRequested (Rowset p_DataUpdateRule)

RECOMMENDATION: Do not override.

Parameters:
   p_DataUpdateRule - The data update rule.

Returns:
   boolean - True if any row of the Data Update Rule Detail (for this entity) has an action not equal to "Do Not Update" (N)
fill

public void fill (Record p_inRec out, boolean p_clearState)

Populates (fills) this entity from the given record.
RECOMMENDATION: Do not override.

Parameters:
  p_inRec - The record structure that will be used to populate this entity
  p_clearState - Whether or not to RESET this entities pre-image

fromXmlNode

public void fromXmlNode (XmlNode parentNode)

Retrieves data to properties based on the fields in the record.
Fields marked ignore are not Processed.
Entity Element Names, wrapping of the ChildEntities or embedding is based on the
entity registry.
RECOMMENDATION: Only override if the xml for this entity has to be custom.

Parameters:
  parentNode - The parent node to add this node to

generateRowset

public Rowset generateRowset ( )

This method generates a rowset of data for the underlying record of this entity based
on the keys of the parent.
It is used as part of the fill process for building out the entity tree.
The method is called as if it were a static method, so it is not associated with any
particular entity data.
The base implementation dynamically matches keys from the parent to the child and
builds the query for the rowset based on that.
RECOMMENDATION: If the key structure between the parent and child does not
match up, or the rowset needs to be built using another method (i.e. a service),
override this method.

Returns:
  Rowset - A rowset populated based on the parent keys, or null if no rows are
  found

generateXSD

public string generateXSD ( )

Generates the xml schema with this node as the root.
RECOMMENDATION: Do not override.
Returns:
  string - The XML Schema as a string

getChildEntity
public ChildEntity getChildEntity (string p_entityID)
  Retrieves the ChildEntity Object based on the entity ID
  RECOMMENDATION: Do not override.

Parameters:
  p_entityID - The entity ID, based on the registry, of the ChildEntity object to retrieve
Returns:
  ChildEntity

getMaxSeqNbrSQL
public number getMaxSeqNbrSQL ( )
  Retrieves the maximum sequence number from the database based on the keystructure of the current entity.
  This method is irrelevant if there are no sequence numbers for the entity.
  RECOMMENDATION: Override if there is a sequence number and it is not related to all the keys.

Returns:
  number - The maximum sequence number found.

hardDelete
public void hardDelete ( MessageLog p_messageLog out)
  Deletes the entity object and all its child entities from the database based on the entityDelete flag.
  RECOMMENDATION: Do not override.

Parameters:
  p_messageLog - The message log

hasEffdt
public boolean hasEffdt ( )
  Returns:
    boolean

hasEffseq
public boolean hasEffseq ()

    Returns:
    boolean

hasKey

public boolean hasKey (string p_keyname)

    Check if the entity has a particular field as a key
    RECOMMENDATION: Do not override.

    Parameters:
    p_keyname - The name of the field to check.
    Returns:
    boolean - True, the field exists and is a key; false, the field is not a key, or
does not exist.

isEqualKeys

public boolean isEqualKeys ( IEntity p_entity)

    Allows the given entity to be compared with this entity for equality. The Base
implementation compares for equality based on the underlying entity record contents.
    RECOMMENDATION: Override this method if you need to do custom equality
checking.

    Parameters:
    p_entity
    Returns:
    boolean

populateAudit

public void populateAudit ( )

    Populates the who column data, this method is invoked prior to save. This allows
handling of the who columns to be more or less dynamic. If there are no who columns
this method does nothing. RECOMMENDATION: Do not override.

prePublish

public void prePublish (Rowset p_changesRS out)

    Uses the workflowPreimage and post images of the entity to build a PSCAMA Rowset
in preparation for publish. The parameter Rowset may be Null. Data should be
appended to this rowset if it is NON-Null. RECOMMENDATION: Override if pre-
publish logic is required.

    Parameters:
    p_changesRS
**preSave**

public void preSave ( )

Allows the opportunity for an entity to perform any pre Save activities. The Base implementation does nothing. RECOMMENDATION: Override this method if the entity needs to do last minute preSave processing.

**preSaveChildren**

public void preSaveChildren ( )

Iterates through the ChildEntityArray and issues a preSave against each ChildEntity, which in turn calls preSave on all of the entities it contains. Base implementation uses the entity registry. RECOMMENDATION: Do not override.

**preSaveFirst**

public void preSaveFirst ( )

Similar to presave, however, this method is called once per childEntity, prior to any other presave activity to allow for presave activity that may act across all entities in the childEntity. The Base implementation does nothing. RECOMMENDATION: Override this method if the entity needs to do last minute preSave processing crossing entity boundaries.

**promote**

abstract public void promote (string p_emplID, MessageLog p_log out)

Moves Data from Staging to Production Records (does not validate or save)

RECOMMENDATION: This must be implemented, but is only used if the entity is staged. In the implementation make sure to call promoteChildren to cause promote to work with the registry.

Parameters:
- p_emplID
- p_log - The MessageLog object to log all errors/warnings to

**publish**

public void publish (Rowset p_changesRS, boolean p_onlineMode)

Takes a PSCAMA based rowset of information to be published during workflow; uses that information to construct a Message and, according to the onlineMode, publishes the Message data in realtime or to the batch delay table. RECOMMENDATION: Override if publish logic is required.

Parameters:
- p_changesRS
- p_onlineMode
retrieveDataRule

public Rowset retrieveDataRule ( )

Uses the entityId and Update Rule Name to read Data Update Rule information
RECOMMENDATION: Do not override.

Returns:
Rowset- The Data Update Rule information for the current entity in a rowset
(Level 0 - SCC_DUR_HDR, Level 1 - SCC_DUR_DTL)

save

public void save ( MessageLog p_messageLog out)

Base implementation, calls presave and issues save for all children then a save for
this entity.
RECOMMENDATION: Do not override.

Parameters:
p_messageLog - The message log

setChildren

public void setChildren ( )

Builds out the ChildEntityArray based on the entity registry.
RECOMMENDATION: Do not override.

setDefault

public void setDefault ( )

Base implementation sets the default value for each field using the Field setDefault
method if the field has not already been marked changed.
RECOMMENDATION: Override if required.

setStageMode

public void setStageMode (boolean p_stageInd)

Sets the stage mode and changes the underlying record to stage/production for this
entity and all children.
This implementation does nothing since stage mode is not required.
RECOMMENDATION: Override if the entity is Staged.

Parameters:
p_stageInd - Set to stage (true) or production (false)
setUpdateRule

public void setUpdateRule (string p_UpdateRule)

Sets a specific update rule
RECOMMENDATION: Do not override.

Parameters:
    p_UpdateRule - The specific update rule

setUpdateRuleByTransName

public void setUpdateRuleByTransName (string p_transName)

Sets the update rule based on the transaction name in the setup table
RECOMMENDATION: Do not override.

Parameters:
    p_transName - The transaction name

setUserContext

public void setUserContext (string p_userContext)

Sets the user context for this entity.
RECOMMENDATION: Do not override.

Parameters:
    p_userContext - The user context to set. An empty string sets the context to %UserId.

toXmlNode

public void toXmlNode (XmlNode parentNode)

Pushes data to the XML structure from the entity.
Fields marked ignore are not Processed.
Entity Element Names, wrapping of the ChildEntities or embedding is based on the entity registry.
The value of properties corresponding to the fields are pushed.
RECOMMENDATION: Only override if the xml for this entity has to be custom.

Parameters:
    parentNode - The parent node of this node

updateChildFields

public void updateChildFields (string propname, array of ChildEntity cea)
Sets the value of a specific property on all ChildEntities to match the current entity
RECOMMENDATION: Do not override.

Parameters:
- propname - The name of the property to update
- cea - The array of child entities (and their children down the tree) to set the property on

validate

public void validate (MessageLog p_messageLog out)
Validates the entity object if entity is not marked for deletion
RECOMMENDATION: Override this method if the entity requires custom validation.
Use commonValidate to call the system edits.
This method does the following standard system edits:
- Required fields are present
- Validates all 1/0 fields contain only a 1 or a 0
- Validates all translate fields have a valid value
- Validates all YesNo fields contain a Y or an N
- Validates all prompt edit fields have a valid value

Parameters:
- p_messageLog - The container for all the messages that will be generated by this method.

validateFirst

public void validateFirst (MessageLog p_messageLog out)
Allows validations to be performed on all entities under a childEntity. This is valuable in doing validation against other entities under the same childEntity.
RECOMMENDATION: Override this method if the entity requires custom validation crossing entity boundaries.

Parameters:
- p_messageLog - The container for all the messages that will be generated by this method.

workflow

public void workflow (boolean p_onlineMode)
Orchestrates the prepublish and publish methods for this entity and all children.
RECOMMENDATION: Do not override.

Parameters:
- p_onlineMode

buildXSDChildren
protected void buildXSDChildren (XmlNode xmlnode)

Iterates through the ChildEntityArray and issues a buildXSD against each ChildEntity, which in turn creates a temporary entity and calls buildXSD. Base implementation uses the entity registry. RECOMMENDATION: Do not override.

Parameters:
xmlnode - The node of the current entity that all children will see as the parent

changeRecord
protected void changeRecord (string p_recname)

Changes the record to the record name specified. RECOMMENDATION: Do not override.

Parameters:
p_recname - The name of the record to have this entity use.

commonValidate
protected void commonValidate (MessageLog p_messageLog out, number p_validateArgs)

Runs the record based validations and outputs results to the log. Validation Arguments:

<table>
<thead>
<tr>
<th>%Edit_DateRange</th>
<th>Reasonable Date Range (Is the date contained within the specified reasonable date range?)</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Edit_OneZero</td>
<td>1/0 (Do all 1/0 fields contain only a 1 or 0?)</td>
</tr>
<tr>
<td>%Edit_PromptTable</td>
<td>Prompt Table (Is field data contained in the specified prompt table?)</td>
</tr>
<tr>
<td>%Edit_Required</td>
<td>Required Field (Do all required fields contain data? For numeric or signed fields, it checks that they do not contain NULL or 0 values.)</td>
</tr>
<tr>
<td>%Edit_TranslateTable</td>
<td>Translate Table (Is field data contained in the specified translate table?)</td>
</tr>
<tr>
<td>%Edit_YesNo</td>
<td>Yes/No (Do all yes/no fields contain only yes or no data?)</td>
</tr>
</tbody>
</table>

Parameters:
p_messageLog - The message log to log any errors to p_validateArgs - Specify the validations to be run in a format like (%Edit_DateRange + %Edit_OneZero + %Edit_Required)

defaultChildren
protected void defaultChildren ( )

Iterates through the ChildEntityArray and issues a setDefault against each ChildEntity, which in turn calls setDefault on all of the entities it contains. Base implementation uses the entity registry. RECOMMENDATION: Do not override.
deleteChildren

protected void deleteChildren (boolean p_entityDelete)

Iterates through the ChildEntityArray and issues a delete against each ChildEntity, which in turn calls delete on all of the entities it contains. Base implementation uses the entity registry.

RECOMMENDATION: Do not override.

Parameters:

  p_entityDelete - True to delete this entity and False to undelete.

demoteChildren

protected void demoteChildren (number p_tempID)

Iterates through the ChildEntityArray and issues a demote against each ChildEntity, which in turn calls demote on all of the entities it contains. Base implementation uses the entity registry.

RECOMMENDATION: Do not override.

Parameters:

  p_tempID - The Tempid to associated with all demoted data.

fillChildren

protected void fillChildren (boolean p_clearState)

Iterates through the ChildEntityArray and issues a fill against each ChildEntity, which in turn calls fill on all of the entities it contains. Base implementation uses the entity registry.

RECOMMENDATION: Do not override.

Parameters:

  p_clearState

fromXmlNodeChildren

protected void fromXmlNodeChildren (XmlNode parentNode)

Iterates through the ChildEntityArray and issues a fromXMLNode against each ChildEntity, which in turn calls fromXMLNode on all of the entities it contains. Base implementation uses the entity registry.

RECOMMENDATION: Do not override.

Parameters:

  parentNode - The node of the current entity that all children will see as the parent
getCSUserDefaults

protected CSUserDefaults getCSUserDefaults ( )
Retrieves the CSUserDefaults
RECOMMENDATION: Do not override.

Returns:
    CSUserDefaults

getInstallationHCM

protected InstallationHCM getInstallationHCM ( )
Retrieves the HCM Installation Defaults
RECOMMENDATION: Do not override.
Returns:
    InstallationHCM

defParent

protected IEntity getParent ( )

Retrieves the parent Entity
RECOMMENDATION: Do not override.

Returns:
    IEntity

getRecordName

protected string getRecordName ( )

Retrieves the name of the current record on this entity.
RECOMMENDATION: Do not override.

Returns:
    string - The record name

hardDeleteChildren

protected void hardDeleteChildren ( MessageLog p_messageLog out)

Iterates through the ChildEntityArray and issues a hardDelete against each
ChildEntity, which in turn calls hardDelete on all of the entities it contains.
Base implementation uses the entity registry.
RECOMMENDATION: Do not override.

Parameters:
    p_messageLog - the message log
promoteChildren

protected void promoteChildren (string p_emplID, MessageLog p_log out)

Iterates through the ChildEntityArray and issues a promote against each ChildEntity, which in turn calls promote on all of the entities it contains. Base implementation uses the entity registry.
RECOMMENDATION: Do not override.

Parameters:
  p_emplID - The EMPLID key to associate with the promoted data.
  p_log

saveChildren

protected void saveChildren (MessageLog p_messageLog out)

Iterates through the ChildEntityArray and issues a Save against each ChildEntity, which in turn calls Save on all of the entities it contains. Base implementation uses the entity registry.
RECOMMENDATION: Do not override.

Parameters:
  p_messageLog - The message log

setData

protected void setData (Record p_dataRec)

Sets the Data record
RECOMMENDATION: Do not override.

Parameters:
  p_dataRec - The record to set data to

setDataPreImage

protected void setDataPreImage (Record p_dataPreImageRec)

sets the Data preimage
RECOMMENDATION: Do not override.

Parameters:
  p_dataPreImageRec - The preimage record to set

setEntityID
protected void setEntityID (string p_entityID)

Sets the entity ID, and retrieves the related information from the registry
RECOMMENDATION: Do not override.

Parameters:
   p_entityID - The ID of the Entity.

setKeyList

protected void setKeyList ( )

Sets the list of key fields to private instance variable keyList
RECOMMENDATION: Do not override.

testForAudit

protected void testForAudit ( )

Checks if the underlying record has the audit fields and sets the property hasAudit.
RECOMMENDATION: Do not override.

toXmlNodeChildren

protected void toXmlNodeChildren (XmlNode parentNode)

Iterates through the ChildEntityArray and issues a toXMLNode against each
ChildEntity, which in turn calls toXMLNode on all of the entities it contains.
Base implementation uses the entity registry.
RECOMMENDATION: Do not override.

Parameters:
   parentNode - The node of the current entity that all children will see as the
   parent

updateKeyFields

protected void updateKeyFields ( )

Updates all the key fields of the child entities that match the key fields on this entity
RECOMMENDATION: Do not override.

validateChildren

protected void validateChildren ( MessageLog p_messageLog out)

Iterates through the ChildEntityArray and issues a validate against each ChildEntity,
which in turn calls validate on all of the entities it contains.
Base implementation uses the entity registry.
RECOMMENDATION: Do not override.
Parameters:
  p_messageLog - The message log

Non-staged entity base class – BasicEntity

SCC_COMMON:ENTITY
Class BasicEntity

SCC_COMMON:ENTITY:AbstractEntity
   |__SCC_COMMON:ENTITY:BasicEntity

This is the base implementation of IEntity to extend for non-staged entities. That is entities that only interact with the production record. Promote and demote have no real purpose here and the methods do nothing.

<table>
<thead>
<tr>
<th>Properties Inherited from SCC_COMMON:ENTITY:AbstractEntity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildEntityArray, data, dataPreimage, DO_SAVE, entityDelete, entityID, entityName, parent, parentCE, prodRecordName, SCC_ROW_ADD_DTTM, SCC_ROW_ADD_OPRID, SCC_ROW_UPD_DTTM, SCC_ROW_UPD_OPRID, selfServiceMode, STAGE_MODE, UPDATE_RULE, USER_CONTEXT, workflowPreimage, XSDFieldList, hasAudit, ignoreFields, RECORD_NAME, XSDNS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public BasicEntity ( IEntity p_parent)</td>
</tr>
<tr>
<td>Base Constructor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void demote (number p_tempID)</td>
</tr>
<tr>
<td>This method does nothing since this entity is not staged, but it has to be implemented</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods Inherited from SCC_COMMON:ENTITY:AbstractEntity</th>
</tr>
</thead>
<tbody>
<tr>
<td>buildXSD, delete, entityFieldUpdateNeeded, entityTypeUpdateNeeded, entityTypeUpdateRequested, entityUpdateRequested, fill, fromXmlNode, generateRowset, generateXSD, getChildEntity, getMaxSeqNbrSQL, hardDelete, hasIfldt, hasEffseq, hasKey, isEqualKeys, populateAudit, prePublish, preSave, preSaveChildren, preSaveFirst, publish, retrieveDataRule, save, setChildren, setDefault, setStageMode, setUpdateRule, setUpdateRuleByTransName, setUserContext, toXmlNode, updateChildFields, validate, validateFirst, workflow, buildXSDChildren, changeRecord, commonValidate, defaultChildren, deleteChildren, demoteChildren, fillChildren, fromXmlNodeChildren, getCSIUserDefaults, getInstallationHCM, getParent, getRecordName, hardDeleteChildren, promoteChildren, saveChildren, setData, setDataPreimage, setEntityID, setKeyList, testForAudit, toXmlNodeChildren, updateKeyFields, validateChildren</td>
</tr>
</tbody>
</table>
Constructor Detail

BasicEntity

public BasicEntity ( IEntity p_parent)

Base Constructor

Parameters:
  p_parent - The parent of this entity, or null if this is the top of the entity tree

Method Detail

demote

public void demote (number p_tempID)

This method does nothing since this entity is not staged, but it has to be implemented

Parameters:
  p_tempID

promote

public void promote (string pemplID, MessageLog p_log out)

This method does nothing since this entity is not staged, but it has to be implemented

Parameters:
  pemplID
  p_log

Staged Entity Base Class – StagedEntity

SCC_COMMON:ENTITY

Class StagedEntity

SCC_COMMON:ENTITY:AbstractEntity

直接已知子类：
Implementing Classes:

- ApplicationFeeTender
- TestScoreComponent

The base implementation to use for entities requiring staging support. The entire entity is designed to allow for data to be moved between staging and production records.

### Property Summary

<table>
<thead>
<tr>
<th>Public</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td><code>DO_SAVE_CHILDREN</code></td>
<td>Specifies if the children should be saved</td>
</tr>
<tr>
<td>string</td>
<td><code>EMPLID</code></td>
<td>The property for the emplid field on production records</td>
</tr>
<tr>
<td>string</td>
<td><code>entityID</code></td>
<td>The entity ID based on the entity registry</td>
</tr>
<tr>
<td>string</td>
<td><code>entityName</code></td>
<td>The entity Name as specified in the registry</td>
</tr>
<tr>
<td>string</td>
<td><code>PROD_RECORD_NAME</code></td>
<td>The Production Record name from the registry</td>
</tr>
<tr>
<td>number</td>
<td><code>SCC_TEMP_ID</code></td>
<td>The property for the tempid field on stage records</td>
</tr>
<tr>
<td>string</td>
<td><code>STAGE_RECORD_NAME</code></td>
<td>The Stage Record name from the registry</td>
</tr>
</tbody>
</table>

### Properties Inherited from SCC_COMMON:ENTITY:AbstractEntity

- `ChildEntityArray`, `data`, `dataPreimage`, `DO_SAVE`, `entityDelete`, `parent`, `parentCE`, `prodRecordName`, `SCC_ROW_ADD_DTTM`, `SCC_ROW_ADD_OPRID`, `SCC_ROW_UPD_DTTM`, `SCC_ROW_UPD_OPRID`, `selfServiceMode`, `STAGE_MODE`, `UPDATE_RULE`, `USER_CONTEXT`, `workflowPreimage`, `XSDFieldList`, `hasAudit`, `ignoreFields`, `RECORD_NAME`, `XSDNS`

### Constructor Summary

- `public StagedEntity ( IEntity p_parent)`
  The Constructor

### Method Summary

- `public void comparePreviousEffdt ()`
  Compares the current record underlying this entity with previous effective dated rows. If no data has changed between this entity and the current effective-dated row, it prevents a save from occurring. This only matters when we are dealing with effective-dated entities.
  RECOMMENDATION: Do not override.

- `public void demote (number p_tempID)`
  Moves data from production records to staging record
RECOMMENDATION: Do not override.

```java
public void fromXMLNode (XmlNode parentNode)

Retrieves data to properties based on the fields in the record. Fields marked ignore are not Processed. Entity Element Names, wrapping of the ChildEntities or embedding is based on the entity registry. Overridden to perform custom processing for SCC_TEMP_ID and EMPLID, depending on production versus staging mode.
RECOMMENDATION: Only override if the xml for this entity has to be custom.
```

RECOMMENDATION: Only override if the xml for this entity has to be custom.

```java
public string getID ()

Retrieves the temp ID
```

```java
public void promote (string p_emplID, MessageLog p_log out)

Moves Data from Staging to Production Records (does not validate or save) RECOMMENDATION: Do not override.
```

```java
public void save (MessageLog p_messageLog out)

Calls presave and issues save for all children then a save for this entity.
RECOMMENDATION: Do not override.
```

RECOMMENDATION: Do not override.

```java
public void setID (string p_ID)

Sets the temp id for this entity
```

RECOMMENDATION: Do not override.

```java
public void setStageMode (boolean p_stageInd)

Sets the stage mode and changes the underlying record to stage/production for this entity and all children
RECOMMENDATION: Do not override.
```

```java
protected string getProdRecordName ()

Retrieves the production record name
```

```java
protected string getStageRecordName ()

Retrieves the stage record name
```

```java
protected void setEntityID (string p_entityID)

Sets the entity ID, and retrieves the related information from the registry
RECOMMENDATION: Do not override.
```

Methods Inherited from SCC_COMMON:ENTITY:AbstractEntity

- buildXSD
- delete
- entityFieldUpdateNeeded
- entityTypeUpdateNeeded
- entityTypeUpdateRequested
- entryUpdateRequested
- fill
- generateRowset
- generateXSD
- getChildEntity
- getMaxSeqNbrSQL
- hardDelete
- hasEffdt
- hasEffseq
- hasKey
- isEqualKeys
- populateAudit
- prePublish
- preSave
- preSaveChildren
- preSaveFirst
- publish
- retrieveDataRule
- setChildren
- setDefault
- setUpdateRule
- setUpdateRuleByTransName
- setUserContext
- toXmlNode
- updateChildFields
- validate
- validateFirst
- workflow
- buildXSDChildren
- changeRecord
- commonValidate
- defaultChildren
- deleteChildren
- demoteChildren
- fillChildren
- fromXmlNodeChildren
- getCSUserDefaults
- getInstallationHCM
- getParent
- getRecordName
- hardDeleteChildren
- promoteChildren
- saveChildren
- setData
- setDataPreImage
- setKeyList
- testForAudit
- toXmlNodeChildren
- updateKeyFields
- validateChildren

Property Detail
DO_SAVE_CHILDREN

public boolean
    Specifies if the children should be saved

EMPLID

public string
    The property for the emplid field on production records

delcomeID

readonly public string
    The entity ID based on the entity registry

dentityName

readonly public string
    The entity Name as specified in the registry

PROD_RECORD_NAME

public string
    The Production Record name from the registry

SCC_TEMP_ID

public number
    The property for the tempid field on stage records

STAGE_RECORD_NAME

public string
    The Stage Record name from the registry

Constructor Detail

StagedEntity

public StagedEntity ( IEntity p_parent)
    The Constructor

Parameters:
p_parent - The parent of this entity, or null if this is the top of the entity tree

Method Detail

**comparePreviousEffdt**

```java
public void comparePreviousEffdt()
```

Compares the current record underlying this entity with previous effective-dated rows. If no data has changed between this entity and the current effective-dated row, it prevents a save from occurring. This only matters when we are dealing with effective-dated entities. RECOMMENDATION: Do not override.

**demote**

```java
public void demote(number p_tempID)
```

Moves data from production records to staging record
RECOMMENDATION: Do not override.

Parameters:
- p_tempID - The tempid to replace the emplid with

**fromXMLNode**

```java
public void fromXMLNode(XmlNode parentNode)
```

Retrieves data to properties based on the fields in the record. Fields marked ignore are not Processed. Entity Element Names, wrapping of the ChildEntities or embedding is based on the entity registry. Overridden to perform custom processing for SCC_TEMP_ID and EMPLID, depending on production versus staging mode. RECOMMENDATION: Only override if the xml for this entity has to be custom.

Parameters:
- parentNode - The parent node to add this node to

**getID**

```java
public string getID()
```

Retrieves the temp ID

Returns:
- string - The tempID

**promote**

```java
public void promote(string p_emplID, MessageLog p_log out)
```

Oracle Proprietary and Confidential
Moves Data from Staging to Production Records (does not validate or save)
RECOMMENDATION: Do not override.

Parameters:
  p_emplID
  p_log - The MessageLog object to log all errors/warnings to

save

public void save ( MessageLog p_messageLog out)

Calls presave and issues save for all children then a save for this entity.
RECOMMENDATION: Do not override.

Parameters:
  p_messageLog - The message log

setID

public void setID (string p_ID)

Sets the temp id for this entity
RECOMMENDATION: Do not override.

Parameters:
  p_ID - The temporary id value to set

setStageMode

public void setStageMode (boolean p_stageInd)

Sets the stage mode and changes the underlying record to stage/production for this
entity and all children
RECOMMENDATION: Do not override.

Parameters:
  p_stageInd - Set to stage (true) or production (false)

getProdRecordName

protected string getProdRecordName ( )

Retrieves the production record name
Returns:
  string

getAddressRecordName

protected string getStageRecordName ( )
Retrieves the stage record name
Returns:
   string

setEntityID

protected void setEntityID (string p_entityID)

Sets the entity ID, and retrieves the related information from the registry
RECOMMENDATION: Do not override.

Parameters:
   p_entityID - The ID of the Entity.

Staged Entity (with Service Driven Production Record) Base Class
– StagedHREntity

SCC_COMMON:ENTITY
Abstract Class StagedHREntity

SCC_COMMON:ENTITY:AbstractEntity

| SCC_COMMON:ENTITY:StagedHREntity Direct Known Subclasses: |
| Disability, LicenseAndCertificate, NationalID, PersonDataUSA, PersDataEffDt, VisaPermitSupport, DriversLicenseType, Citizenship, Constituent, Phone, Diversity, CitizenshipPassport, EmergencyContact, EmergencyPhone, Publications, Address, DiversEthnic, PersonCAN, DriversLicense, Names, EmailAddresses, Membership, VisaPermitData, Language |

The Base abstract implementation class for all Campus HCM entities that require Staging support.
To interact with HCM Entities we expect to use web services, the two key methods for interaction are as follows:

UpdateDeleteHCM - Where we call the service to update/add HCM data.
generateRowset - Which needs to be overridden to allow for access to get data from a service, instead of directly accessing the production record provided.

<table>
<thead>
<tr>
<th>Property Summary</th>
</tr>
</thead>
</table>
| public string EMPLID
   The property for the emplid field on production records |
| public string entityID readonly
   The entity ID based on the entity registry |
| public string entityName readonly
   The entity Name as specified in the registry |
### Properties Inherited from SCC_COMMON:ENTITY:AbstractEntity

- ChildEntityArray
- data
- dataPreimage
- DO_SAVE
- entityDelete
- parent
- parentCE
- prodRecordName
- SCC_ROW_ADD_DTTM
- SCC_ROW_ADD_OPRID
- SCC_ROW_UPD_DTTM
- SCC_ROW_UPD_OPRID
- selfServiceMode
- STAGE_MODE
- UPDATE_RULE
- USER CONTEXT
- workflowPreimage
- XSDFieldList
- hasAudit
- ignoreFields
- RECORD_NAME
- XSDNS

### Constructor Summary

#### public

**public** Sting **PROD_RECORD_NAME**

The Production Record name from the registry

**public** Number **SCC_TEMP_ID**

The property for the tempid field on stage records

**public** Sting **STAGE_RECORD_NAME**

The Stage Record name from the registry

### Constructor Summary

**public** StagedHREntity( IEntity p_parent)

The Constructor

### Method Summary

#### public void **dematic** (number p_tempID)

Moves data from production records to staging record

RECOMMENDATION: Do not override.

#### public void **fromXmlNode** (XmlNode parentNode)

Retrieves data to properties based on the fields in the record. Fields marked ignore are not Processed. Entity Element Names, wrapping of the ChildEntities or embedding is based on the entity registry. Overridden to perform custom processing for SCC_TEMP_ID and EMPLID, depending on production vs staging mode. RECOMMENDATION: Only override if the xml for this entity has to be custom.

#### public baseType **getHRType** (string p_EMPLID)

Builds the HCR_PERSON_TYPES object based on the data in the current entity using the emplid provided

#### public string **getID** ()

Retrieves the temp ID

#### public void **hardDelete** (MessageLog p_messageLog out)

Deletes the entity object and all its child entities from the database, based on the entityDelete flag. RECOMMENDATION: Do not override.

#### public void **promote** (string p_emplID, MessageLog p_log out)

Moves Data from Staging to Production Records (does not validate or save) RECOMMENDATION: Do not override.

#### public void **save** (MessageLog p_messageLog out)
Calls presave and issues save for all children, then a save for this entity. On save for production it invokes updateDeleteHcm to call the HCM Web Service. 
RECOMMENDATION: Do not override.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void setID (string p_ID)</td>
<td>Sets the temp id for this entity. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td>public void setStageMode (boolean p_stageInd)</td>
<td>Sets the stage mode and changes the underlying record to stage/production for this entity and all children. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td>protected ServiceManager getServiceManager ( )</td>
<td>Returns the HCM ServiceManager object.</td>
</tr>
<tr>
<td>protected void setEntityID (string p_entityID)</td>
<td>Sets the entity ID and retrieves the related information from the registry. RECOMMENDATION: Do not override.</td>
</tr>
<tr>
<td>abstract protected void updateDeleteHCM (boolean p_deleteFlag)</td>
<td>This method is called from save. It should perform the update for production data by calling the HCM Web Service.</td>
</tr>
</tbody>
</table>

Methods Inherited from SCC_COMMON:ENTITY:AbstractEntity:
buildXSD, delete, entityFieldUpdateNeeded, entityTypeUpdateNeeded, entityTypeUpdateRequested, entityUpdateRequested, fill, generateRowset, generateXSD, getChildEntity, getMaxSeqNbrSQL, hasEffdt, hasEffseq, hasKey, isEqualKeys, populateAudit, prePublish, preSave, preSaveChildren, preSaveFirst, publish, retrieveDataRule, setChildren, setDefault, setUpdateRule, setUpdateRuleByTransName, setUserContext, toXmlNode, updateChildFields, validate, validateFirst, workflow, buildXSDChildren, changeRecord, commonValidate, defaultChildren, deleteChildren, demoteChildren, filChildren, fromXmlNodeChildren, getCSUserDefaults, getInstallationHCM, getParent, getRecordName, hardDeleteChildren, promoteChildren, saveChildren, setData, setDataPreImage, setKeyList, testForAudit, toXmlNodeChildren, updateKeyFields, validateChildren

Property Detail

**EMPLID**

public string
The property for the emplid field on production records.

**entityID**

readonly public string
The entity ID based on the entity registry.

**entityName**
readonly public string
    The entity Name as specified in the registry

PROD_RECORD_NAME

public string
    The Production Record name from the registry

SCC_TEMP_ID

public number
    The property for the tempid field on stage records

STAGE_RECORD_NAME

public string
    The Stage Record name from the registry

Constructor Detail

StagedHREntity

public StagedHREntity ( IEntity p_parent)

    The Constructor

    Parameters:
    
    p_parent - The parent of this entity, or null if this is the top of the entity tree

Method Detail

demote

public void demote (number p_tempID)

    Moves data from production records to staging record
    RECOMMENDATION: Do not override.

    Parameters:
    
    p_tempID - The tempid to replace the emplid with

fromXmlNode

public void fromXmlNode (XmlNode parentNode)

    Retrieves data to properties based on the fields in the record.
Fields marked ignore are not Processed.
Entity Element Names, wrapping of the ChildEntities or embedding is based on the
entity registry.
Overridden to perform custom processing for SCC_TEMP_ID and EMPLID, depending
on production versus staging mode.
RECOMMENDATION: Only override if the xml for this entity has to be custom.

Parameters:
   parentNode - The parent node to add this node to

---

getHRTYPE

public baseType getHRTYPE (string p_EMPLID)

Builds the HCR_PERSON_TYPES object based on the data in the current entity using
the emplid provided

Parameters:
   p_EMPLID - The emplid to use for the PERSON_TYPE
Returns:
   baseType

---

getID

public string getID ( )

Retrieves the temp ID

Returns:
   string - The tempID

---

hardDelete

public void hardDelete ( MessageLog p_messageLog out)

Deletes the entity object and all its child entities from the database based on the
entityDelete flag.
RECOMMENDATION: Do not override.

Parameters:
   p_messageLog - The message log

---

promote

public void promote (string p_emplID, MessageLog p_log out)

Moves Data from Staging to Production Records (does not validate or save)
RECOMMENDATION: Do not override.

Parameters:
p_emplID
p_log - The MessageLog object to log all errors/warnings to

save
public void save (MessageLog p_messageLog out)

Calls presave and issues save for all children then a save for this entity.
On save for production it invokes updateDeleteHCM to call the HCM Web Service.
RECOMMENDATION: Do not override.

Parameters:
  p_messageLog - The message log

setID
public void setID (string p_ID)

Sets the temp id for this entity
RECOMMENDATION: Do not override.

Parameters:
  p_ID - The temporary id value to set

setStageMode
public void setStageMode (boolean p_stageInd)

Sets the stage mode and changes the underlying record to stage/production for this
entity and all children
RECOMMENDATION: Do not override.

Parameters:
  p_stageInd - Set to stage (true) or production (false)

getServiceManager
protected ServiceManager getServiceManager ( )

Returns the HCM ServiceManager object

Returns:
  ServiceManager

setEntityID
protected void setEntityID (string p_entityID)
Sets the entity ID and retrieves the related information from the registry
RECOMMENDATION: Do not override.

Parameters:
  p_entityID - The ID of the Entity.

updateDeleteHCM

abstract protected void updateDeleteHCM (boolean p_deleteFlag)

This method is called from save, it should perform the update for production data by calling the HCM Web Service

Parameters:
  p_deleteFlag - Specifies if the entity should be deleted

Class Relating Parent and Child Entities – ChildEntity

SCC_COMMON:ENTITY

Class ChildEntity

This object exists between a parent entity and all the children that implement a specific entity type.

<table>
<thead>
<tr>
<th>Property Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public Array of IEntity</strong> childEntities</td>
</tr>
<tr>
<td>An array of all child entities of a specific type</td>
</tr>
<tr>
<td><strong>public string</strong> className</td>
</tr>
<tr>
<td>The name of the appclass the entities under this childEntity implement</td>
</tr>
<tr>
<td><strong>public string</strong> elementName</td>
</tr>
<tr>
<td>The element name to use for each entity</td>
</tr>
<tr>
<td><strong>public boolean</strong> Embed</td>
</tr>
<tr>
<td>Specifies if the Entities this child contains have been marked for embedding in</td>
</tr>
<tr>
<td>the parent entity</td>
</tr>
<tr>
<td><strong>public string</strong> Encapsulate</td>
</tr>
<tr>
<td>The name of the tag to use to encapsulate all entities in this childEntity object</td>
</tr>
<tr>
<td><strong>public string</strong> EntityID</td>
</tr>
<tr>
<td>The Entity ID for the entities under this childEntity</td>
</tr>
<tr>
<td><strong>public string</strong> EntityName</td>
</tr>
<tr>
<td>The Entity Name for the entities under this childEntity</td>
</tr>
<tr>
<td><strong>public string</strong> MaxCount</td>
</tr>
<tr>
<td>The maximum number of entities of this type allowed based on the registry</td>
</tr>
<tr>
<td>public number</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>The current maximum sequence number, if this entity has a key that is a sequence number</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public string</th>
<th>MinCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>The minimum number of entities of this type allowed based on the registry</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public float</th>
<th>parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>The parent of this ChildEntity</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public string</th>
<th>PROD_RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>The name of the production record the entities under this childEntity implement</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public boolean</th>
<th>STAGE_MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>The current stage state of the entities, where true=staged, false=production</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public string</th>
<th>STAGE_RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>The name of the stage record the entities under this childEntity implement</td>
<td></td>
</tr>
</tbody>
</table>

### Constructor Summary

| public ChildEntity () |

### Method Summary

<table>
<thead>
<tr>
<th>public void</th>
<th>buildXSD (XmlNode xmlnode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The method to build the schema for this entity, works in conjunction with the method of the same name on the entity</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public void</th>
<th>checkForDupes (MessageLog p_messageLog out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checks for entities with duplicate keys in the childEntities array</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public void</th>
<th>clearEntities ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resets the childEntities array to empty</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public IEntity</th>
<th>createEntity ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creates an instance of the entity based on the className</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public void</th>
<th>default ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invokes setDefault on all entities in childEntities</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public void</th>
<th>delete (boolean p_entityDelete)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invokes delete on all entities in childEntities</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public void</th>
<th>demote (number p_tempID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invokes demote on all entities in childEntities</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public void</th>
<th>fill (boolean p_clearstate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invokes fill on all entities in childEntities</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public void</th>
<th>fromXMLNode (XmlNode parentnode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invokes fromXMLNode on all entities in childEntities</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public array of IEntity</th>
<th>getEntitiesByProperties (array of PStruct pstructs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns all entities in the childEntities array where the properties provided match</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>public array of IEntity</th>
<th>getEntitiesByProperty (PStruct pstruct)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns all entities in the childEntities array where the property provided matches</td>
<td></td>
</tr>
</tbody>
</table>
### Public Methods

**public IEntity**

**getEntityByProperties** (array of PStruct pstructs)

Returns a specific entity in the childEntities array where all properties provided match.

**public IEntity**

**getEntityByPropertiesEffdt** (array of PStruct pstructs, date asOfDate)

Returns a specific entity in the childEntities array where all properties provided match and the entity is the current entity as of a specific effective date.

**public IEntity**

**getEntityByProperty** (PStruct pstruct)

Returns a specific entity in the childEntities array where a property matches that requested.

**public number**

**getNextSeqNbr** (string fieldname, array of IEntity entities)

Provides the next sequence number for a specified fieldname out of a specific array of entities. Setting the array allows the entities to be a subset of childEntities.

**public Rowset**

**getRowset** ()

Builds an empty rowset of stage or production based on the STAGE_MODE.

**public void**

**hardDelete** (MessageLog p_messageLog out)

Invokes hardDelete on all entities in childEntities.

**public void**

**preSave** ()

Invokes presaveFirst on the first entity in childEntities and presave on all entities in childEntities.

**public void**

**promote** (string p_emplID, MessageLog p_messageLog out)

Invokes promote on all entities in childEntities.

**public void**

**save** (MessageLog p_messageLog out)

Invokes save on all entities in childEntities.

**public void**

**toXMLNode** (XmlNode parentnode)

Invokes toXMLNode on all entities in childEntities.

**public void**

**validate** (MessageLog p_messageLog out)

Invokes validate on all entities in childEntities.

**public void**

**workflow** (boolean p_onlineMode)

Invokes all the workflow methods in childEntities.

### Property Detail

**childEntities**

**public Array of IEntity**

An array of all child entities of a specific type.

**className**

**public string**

The name of the appclass the entities under this childEntity implement.

**elementName**
public string
The element name to use for each entity

---

**Embed**

public boolean
Specifies if the Entities this child contains have been marked for embedding in the parent entity

---

**Encapsulate**

public string
The name of the tag to use to encapsulate all entities in this childEntity object

---

**EntityID**

public string
The Entity ID for the entities under this childEntity

---

**EntityName**

public string
The Entity Name for the entities under this childEntity

---

**MaxCount**

public string
The maximum number of entities of this type allowed based on the registry

---

**MaxSeqNbr**

public number
The current maximum sequence number, if this entity has a key that is a sequence number

---

**MinCount**

public string
The minimum number of entities of this type allowed based on the registry

---

**parent**

public IEntity
The parent of this ChildEntity
**PROD_RECORD**

```java
public string
    The name of the production record the entities under this childEntity implement
```

**STAGE_MODE**

```java
public boolean
    The current stage state of the entities, where true=staged, false=production
```

**STAGE_RECORD**

```java
public string
    The name of the stage record the entities under this childEntity implement
```

---

**Constructor Detail**

**ChildEntity**

```java
public ChildEntity ()
```

---

**Method Detail**

**buildXSD**

```java
public void buildXSD (XmlNode xmlnode)
    The method to build the schema for this entity which works in conjunction with the method of the same name on the entity

Parameters:
    xmlnode - The XmlNode object from the parent to add the schema elements to
```

**checkForDupes**

```java
public void checkForDupes ( MessageLog p_messageLog out)
    Checks for entities with duplicate keys in the childEntities array

Parameters:
    p_messageLog
```

**clearEntities**

```java
public void clearEntities ( )
```
Resets the childEntities array to empty

createEntity

public IEntity createEntity ( )

create an instance of the entity based on the className

Returns:

IEntity

default

public void default ( )

Invokes setDefault on all entities in childEntities

delete

public void delete (boolean p_entityDelete)

Invokes delete on all entities in childEntities

Parameters:

p_entityDelete - The delete state passed from the parent to the child

demote

public void demote (number p_tempID)

Invokes demote on all entities in childEntities

Parameters:

p_tempID - The tempID to set on the records when the data is demoted

fill

public void fill (boolean p_clearstate)

Invokes fill on all entities in childEntities

Parameters:

p_clearstate - Specifies if the preImage should be set

fromXMLNode

public void fromXMLNode (XmlNode parentnode)

Invokes fromXMLNode on all entities in childEntities

Parameters:
getEntitiesByProperties

public array of IEntity getEntitiesByProperties ( array of PStruct pstructs)

Returns all entities in the childEntities array where the properties provided match

Parameters:
   pstructs
Returns:
   array of IEntity

getEntitiesByProperty

public array of IEntity getEntitiesByProperty ( PStruct pstruct)

Returns all entities in the childEntities array where the property provided matches

Parameters:
   pstruct - An object that provides the property name and value to search for
Returns:
   array of IEntity

gentityByProperties

public IEntity getEntityByProperties ( array of PStruct pstructs)

Returns a specific entity in the childEntities array where all properties provided match

Parameters:
   pstructs
Returns:
   IEntity

gentityByPropertiesEffdt

public IEntity getEntityByPropertiesEffdt ( array of PStruct pstructs, date asOfDate)

Returns a specific entity in the childEntities array where all properties provided match and the entity is the current entity as of a specific effective date

Parameters:
   pstructs
   asOfDate - The date to check against the effective date
Returns:
   IEntity
getEntityByProperty

public IEntity getEntityByProperty ( PStruct pstruct)

Returns a specific entity in the childEntities array where a property matches that requested

Parameters:
PStruct - An object that provides the property name and value to search for

Returns:
IEntity

getNextSeqNbr

public number getNextSeqNbr (string fieldname, array of IEntity entities)

Provides the next sequence number for a specified fieldname out of a specific array of entities
Setting the array allows the entities to be a subset of childEntities

Parameters:
fieldname - The field name of the sequence number
entities - The array of entities to search through to figure out the max sequence number

Returns:
number - The next available sequence number

getRowset

public Rowset getRowset ( )

Builds an empty rowset of stage or production based on the STAGE_MODE

Returns:
Rowset - The rowset generated

hardDelete

public void hardDelete ( MessageLog p_messageLog out)

Invokes hardDelete on all entities in childEntities

Parameters:
p_messageLog - The messageLog object passed from the parent to the child to log any errors or warnings

preSave

public void preSave ( )
Invokes presaveFirst on the first entity in childEntities and presave on all entities in childEntities

promote

public void promote (string p_emplID, MessageLog p_messageLog out)
Invokes promote on all entities in childEntities

Parameters:
- p_emplID - The employee ID to set on the records when the data is promoted
- p_messageLog - The messageLog object passed from the parent to the child to log any errors or warnings

save

public void save (MessageLog p_messageLog out)
Invokes save on all entities in childEntities

Parameters:
- p_messageLog - The messageLog object passed from the parent to the child to log any errors or warnings

toXMLNode

public void toXMLNode (XmlNode parentnode)
Invokes toXMLNode on all entities in childEntities

Parameters:
- parentnode - The xml node that entities in this childEntity should be added to

validate

public void validate (MessageLog p_messageLog out)
Invokes validate on all entities in childEntities

Parameters:
- p_messageLog - The messageLog object passed from the parent to the child to log any errors or warnings

workflow

public void workflow (boolean p_onlineMode)
Invokes all the workflow methods in childEntities

Parameters:
- p_onlineMode - The online mode
Message Log – MessageLog

SCC_SL.Transaction:INTFC

Class MessageLog

MessageLog
The log of all messages generated by Transaction processing. The Transaction should use this container class to record any non-fatal information, warning or error messages generated during processing.
This information, once captured, will be centrally logged by the Constituent Staging framework. Severity codes are as follows:
3 - Reconcile Error
2 - Error
1 - Warning
0 - Information

<table>
<thead>
<tr>
<th>Property Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean</td>
</tr>
<tr>
<td>isError</td>
</tr>
<tr>
<td>readonly</td>
</tr>
<tr>
<td>public boolean</td>
</tr>
<tr>
<td>isInformation</td>
</tr>
<tr>
<td>readonly</td>
</tr>
<tr>
<td>public boolean</td>
</tr>
<tr>
<td>isReconcileError</td>
</tr>
<tr>
<td>readonly</td>
</tr>
<tr>
<td>public boolean</td>
</tr>
<tr>
<td>isWarning</td>
</tr>
<tr>
<td>readonly</td>
</tr>
<tr>
<td>public string</td>
</tr>
<tr>
<td>messageContext</td>
</tr>
<tr>
<td>public string</td>
</tr>
<tr>
<td>transacName</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
</tr>
<tr>
<td>MessageLog</td>
</tr>
<tr>
<td>(string p_transactionName, number p_tempConstID, string p_messageContext)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void</td>
</tr>
<tr>
<td>append</td>
</tr>
<tr>
<td>(MessageLog p_messageLog)</td>
</tr>
<tr>
<td>public void</td>
</tr>
<tr>
<td>clear</td>
</tr>
<tr>
<td>()</td>
</tr>
<tr>
<td>public string</td>
</tr>
<tr>
<td>getContext</td>
</tr>
<tr>
<td>(number p_index)</td>
</tr>
<tr>
<td>public number</td>
</tr>
<tr>
<td>getMsgID</td>
</tr>
<tr>
<td>(number p_index)</td>
</tr>
</tbody>
</table>
public number getMsgSet (number p_index)

public string getParm1 (number p_index)

public string getParm2 (number p_index)

public string getParm3 (number p_index)

public string getParm4 (number p_index)

public string getParm5 (number p_index)

public number getSeverity (number p_index)

public number length ()

public string read (number p_index)

public void save ()

public void write (number p_msgSet, number p_msgID, number p_severity, string p_parm1, string p_parm2, string p_parm3, string p_parm4, string p_parm5)

public void writeContext (number p_msgSet, number p_msgID, number p_severity, string p_parm1, string p_parm2, string p_parm3, string p_parm4, string p_parm5, string p_messageContext)

**Property Detail**

**isError**
readonly public boolean

**isInformation**
readonly public boolean

**isReconcileError**
readonly public boolean

**isWarning**
readonly public boolean

**messageContext**
readonly public boolean
public string transacName

public string

Constructor Detail

MessageLog

public MessageLog (string p_transactionName, number p_tempConstID, string p_messageContext)

Parameters:
  p_transactionName – The name of the Transaction performing the logging
  p_tempConstID – The Temporary Constituent ID for which the entry belongs
  p_messageContext – The Context of the Log Entry

Method Detail

append

public void append (MessageLog p_messageLog)

Parameters:
  p_messageLog

clear

public void clear ( )

getContext

public string getContext (number p_index)

Parameters:
  p_index

Returns:
  string

getMsgID

public number getMsgID (number p_index)

Parameters:
getMsgSet

public number getMsgSet (number p_index)

Parameters:
    p_index
Returns:
    number

getParm1

public string getParm1 (number p_index)

Parameters:
    p_index
Returns:
    string

getParm2

public string getParm2 (number p_index)

Parameters:
    p_index
Returns:
    string

getParm3

public string getParm3 (number p_index)

Parameters:
    p_index
Returns:
    string

getParm4

public string getParm4 (number p_index)

Parameters:
    p_index
Returns:
    string
getParm5

public string getParm5 (number p_index)

Parameters:
    p_index
Returns:
    string

getSeverity

public number getSeverity (number p_index)

Parameters:
    p_index
Returns:
    number

length

public number length ( )

Returns:
    number

read

public string read (number p_index)

Parameters:
    p_index
Returns:
    string

save

public void save ( )

write

public void write (number p_msgSet, number p_msgID, number p_severity, string p_parm1, string p_parm2, string p_parm3, string p_parm4, string p_parm5)

Parameters:
    p_msgSet
writeContext

public void writeContext (number p_msgSet, number p_msgID, number p_severity, string p_parm1, string p_parm2, string p_parm3, string p_parm4, string p_parm5, string p_messageContext)

Parameters:

  p_msgSet
  p_msgID
  p_severity
  p_parm1
  p_parm2
  p_parm3
  p_parm4
  p_parm5
  p_messageContext