ConcourseSuite Support
Technical Documentation

Last Modified: 2014-06-03 17:31:48.019

Technical Documentation

The technical documentation is divided into the following general topics:

- Server System Requirements
- Application Architecture
- Application Components
- Modules
- Implementation Guide
    - Roles
        - Business Management Role
        - Installation Role
        - Configuration and Customization Role
- Phases
    - Analysis
    - Planning
    - Installation
    - Deployment
        - Training
        - Importing Data
- Customization
    - Data Configuration
    - Rules Engine Customization
    - Customizing the Look & Feel
    - Writing Reports
    - Development
- Post-Implementation
- Development Process
- Developer Tools
- Developer References
- Code Structure
- Installation
- Build Process from Source
- Developing a ConcourseSuite Module
    - ConcourseSuite Framework Model
    - Code Action Classes
    - Configure cfs-config.xml
    - Configure cfs-modules.xml

System Requirements

Server software and hardware requirements.

Software

Concourse Suite works on a number of platforms. This means that you can choose what works best for your organization.

Operating Systems
Linux
Mac OSX
Microsoft Windows
Solaris

Database Servers	
PostgreSQL	9.0+

PostgreSQL is the ConcourseSuite developer's reference database. Please check the individual release notes to ensure compatibility with your chosen database.

Application Server + Java VM
Apache Tomcat 7 + Java 7

Apache Tomcat is the ConcourseSuite developer's reference web application server. Production uses Apache Tomcat 7 and Oracle Java 7.

Server Integration	
SMTP Server	Allows the CRM to send email
Fax Server	Allows the CRM to send faxes using HylaFAX
XMPP Server	Allows the CRM to send instant messages to users, Wildfire is recommended
Asterisk Server	Allows the CRM to integrate with incoming and outgoing calls using Asterisk
LDAP Server	Allows the CRM to validate user access

Hardware

Two cores (of any size) are recommended, as ConcourseSuite makes use of multi-threaded Java operations.

ConcourseSuite is lightweight. Any server you can buy today is generally more than you'll need. However, configure it based on the needs of your organization. If you will be supporting many simultaneous users, configure more and faster CPUs and more memory. If 100% uptime is required, a second spare server is recommended. If your organization plans to save lots of documents, increase the drive space. A single server can be used for the web server, database server, and mail server.

A sensible server for real use might be:

A single dual core processor

2GB RAM

Disk to suit the purpose.

This configuration is enough for all but the most serious applications and will serve over 1000 users.

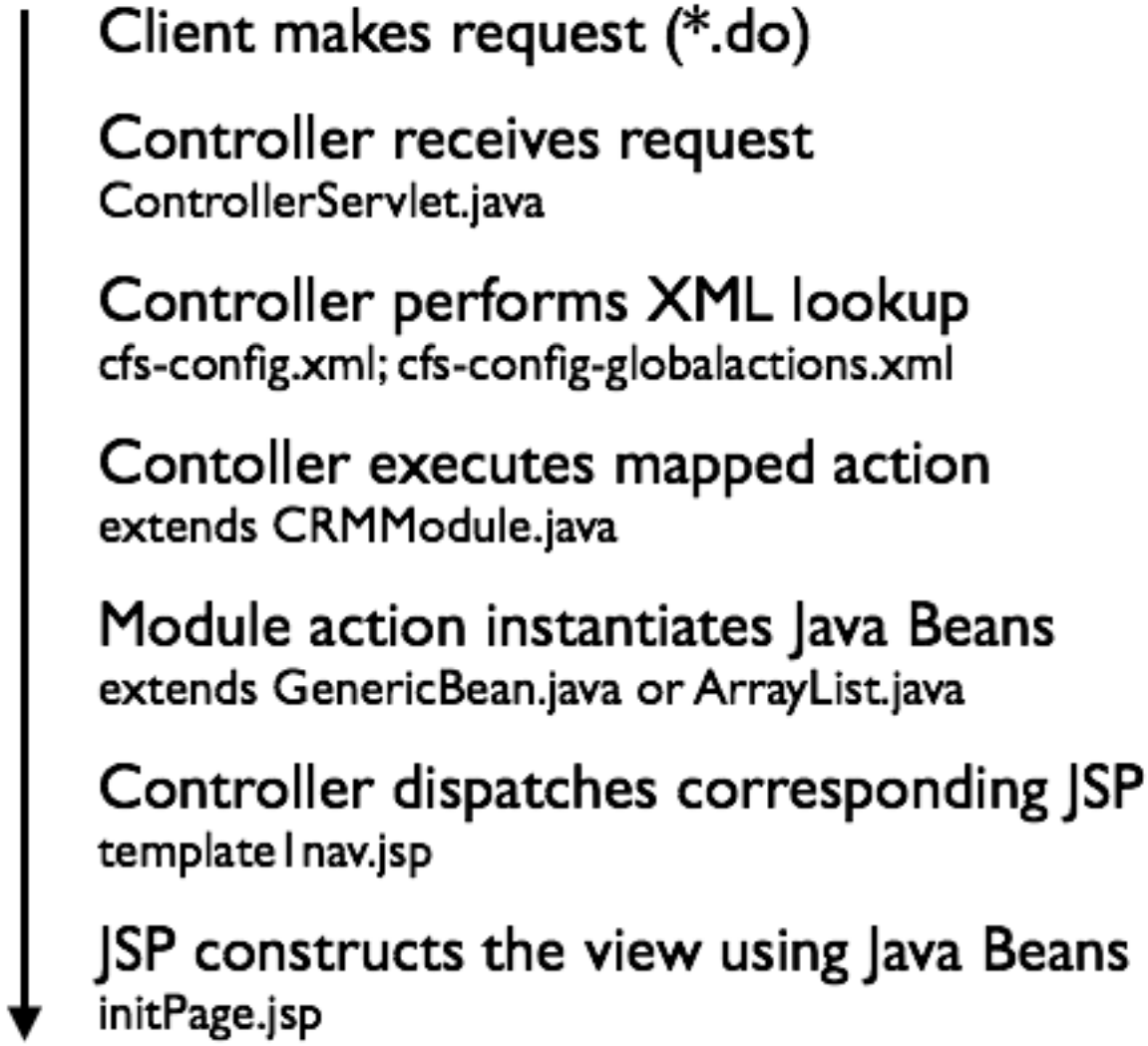
Application Architecture

Client/Server Interaction

Historically, Concourse Suite chose the Theseus MVC framework back in 2000, at a time when Struts was in early development. Theseus is lightweight, works similar to Struts, and allowed the Concourse Suite team to add value around the basic MVC functionality. The team took into consideration application security, performance, layouts, and reusable utilities to form the Concourse Suite framework.

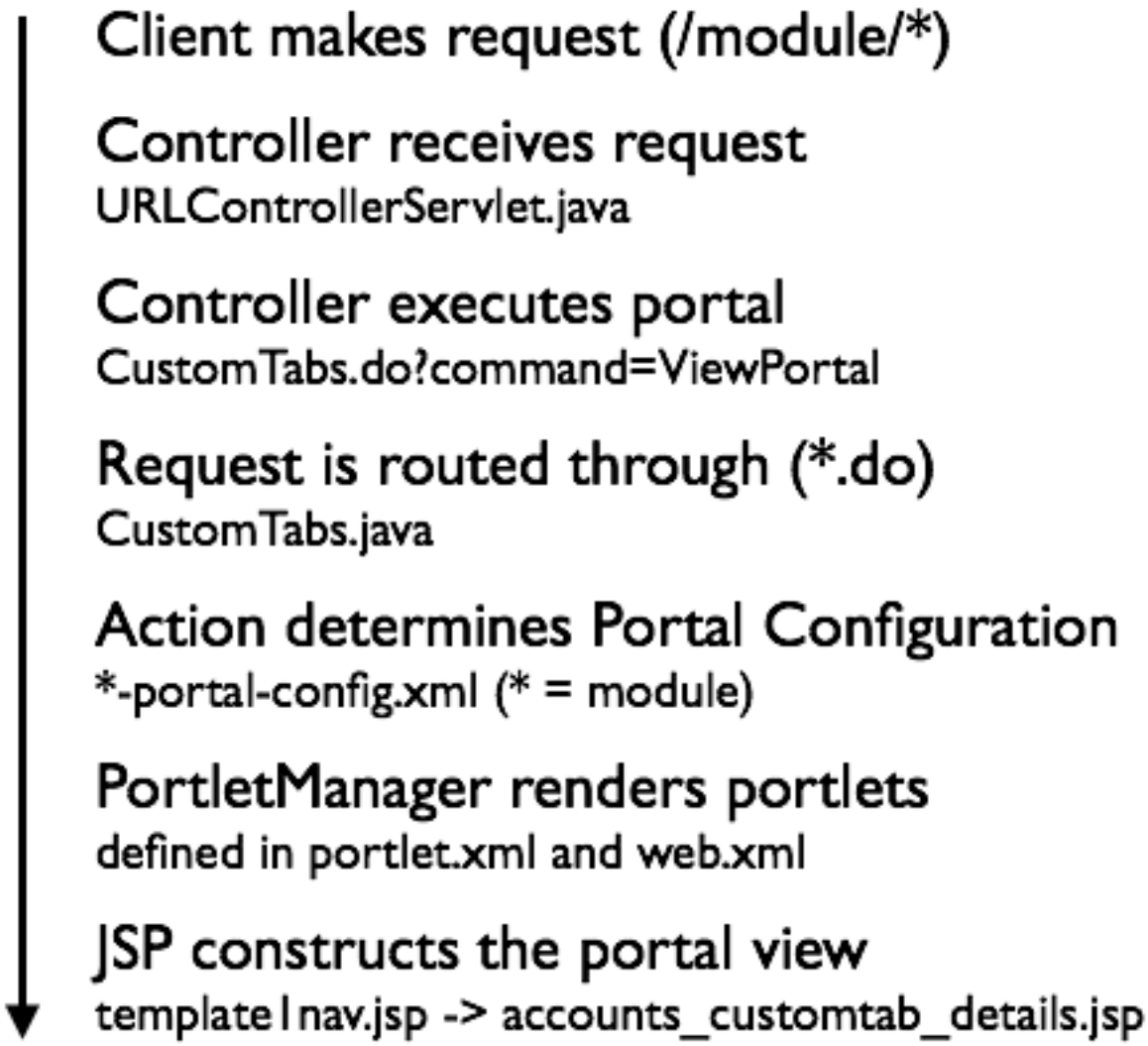
The Concourse Suite framework uses the Model-View-Controller (MVC) web design paradigm. This methodology separates the application business logic and the presentation layer to increase reusability and minimize development time.

Starting with CRM 6, and receiving major improvements in CRM 7, URLs can be mapped to embedded portals to display pages with portlets. The flow is as follows:



In the Concourse Suite framework implementation, a single Servlet receives every web request. This Controller Servlet uses a customizable map to forward each request to a specific Java method, called an Action Class. Each Action Class has a specific task, based on an action the user is trying to complete. When the Action Class is finished, it tells the Servlet Controller and the Controller forwards the request, as well as any data that was prepared, to a JSP. The JSP uses the JavaBeans that were prepared and constructs an HTML page.

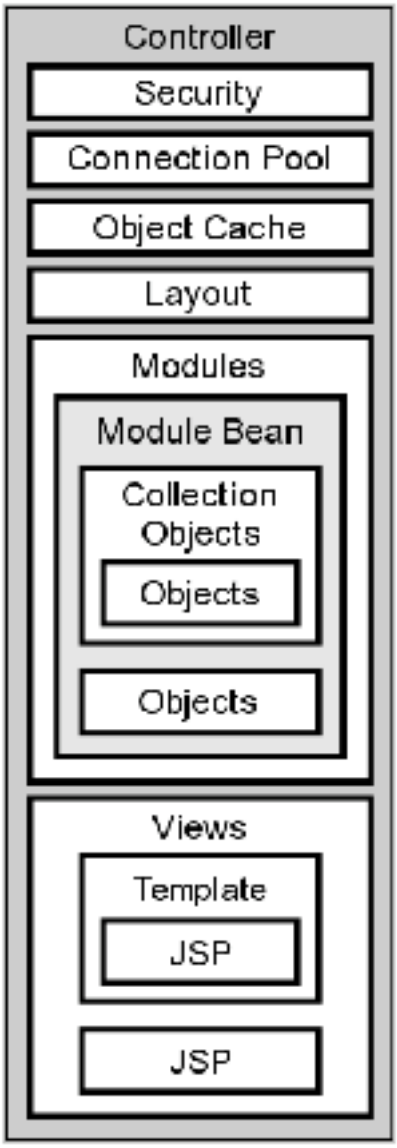
The servlet controller is responsible for processing HTTP requests, executing business logic, and forwarding the result to a JSP for presentation to the client. The resulting JSP should not contain any business logic.



The controller actually maintains and executes much more than just simple requests and responses. Review the following figure for an overview of the framework components. These components, including page layouts, hooks, security, utilities and more are discussed later.

Understanding the basic operation of the servlet controller is important. The servlet controller processes all HTTP requests made by a client in which the URL ends in ".do". For example, the following URL specifies the “Login” action: "http://127.0.0.1/Login.do".

The servlet controller maps this requested action (the part before the .do) to a Java class as defined in cfs-config.xml. So, "Login" means the controller will execute the corresponding Java class, Login.java.



If a Java Bean is specified in the web application configuration, and the HTTP request includes an “auto-populate” parameter, then the controller instantiates the Java class and dynamically populates the object with corresponding values in the request. The resulting object is intended to be used by the action class or sub-classes for the remainder of the request. This is important because the developer does not have to use the typical:

```
someObject.setSomeProperty(request.getParameter(“someParameter”))
```

since auto-populate will map HTML name-value pairs to the specified JavaBean.

The action class then processes the request and returns a single status string to the controller depending on the successful completion of the action.

The servlet controller then maps the action status string to a JSP file for dispatching to.

If a page layout or page template is specified in the mapping, then the layout and template page are dispatched instead, which will then dispatch the final JSP. This allows for menus, static header and static footer content to be re-used.

A developer will typically code an Action Class for each action the user is trying to perform. To reduce duplicate code, actions can be chained and sub-actions can be re-used by more than one action.

The dispatched JSPs contain the HTML for the client. JSPs can be stand-alone files or they can be included in other JSPs for reusability.

JSPs use the Java Beans that were instantiated during the request.

The controller implements various hooks so that the framework can be extended and customized. Hooks add additional functionality during servlet initialization, request processing, reloading, and shutdown. For example, an initialization hook might initialize a database connection pool and object cache, while a request hook might implement security functionality.

Database Portability

The framework is designed to work with commercial database servers, as well as open-source servers. To maintain compatibility and portability, Concourse Suite relies little on stored procedures and more on database independent SQL-92 and SQL-99 syntax.

Database specific code can be used, however, to increase performance while maintaining portability by wrapping database specific code in database-detection methods.

Servlet Container Portability

The Concourse Suite application does not require any unique setup at the container level. However, you might install an SSL certificate or make virtual host settings. The application framework is just beginning to be integrated with containers besides Tomcat. The only modification that remains is for the application to retrieve information from a resource, instead of at the file system level.

Java Coding Conventions by Sun/Oracle

Coding conventions allow for greater understanding of code and easier maintenance. The framework code has been developed with Sun Coding Conventions in mind. This means that code is documented and formatted accordingly, usually automatically by the IDE.

http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html

Web browser compatibility

Since Concourse Suite is a web application, we've found it important to be compatible with as many web browsers as possible.

The framework is designed to work with Internet Explorer, Mozilla Firefox, Google Chrome, and Apple Safari to ensure compatibility with a large platform-independent user base. At the browser level, JavaScript is used for validation and dynamic interaction with page elements. Flash and Java Applets are not currently used.

Browser detection at the framework level can be used in both the business logic and the presentation layer to enhance user experience.

Currently Supported Browsers
Apple Safari
Google Chrome
Internet Explorer 8+
Mozilla Firefox

Application Components

While most MVC implementations provide action class caches, auto-population of objects, and tag libraries, the Concursive framework goes beyond the basics and adds real world features that are used in web-based applications. The following list includes features that have been added to the framework:

Security

1. SSL logins – providing a consistent entry-point into the application
2. Single session sign-on per user name – if a user is already logged in, a second login prompts if the user should cancel the other session and continue
3. Client browser auto-logout when server session expires
4. Roles and permissions – each user can have a single role in which permissions are granted
5. User hierarchy – each user can report to another user
6. Action level permissions – restricts module usage based on user role permissions
7. Record level permissions – restricts record usage based on record ownership and user hierarchy
8. Protection from concurrent updates to the same record – users are notified when another user has already updated a record in which they are trying to update
9. One-way encrypted passwords stored in the database
10. Private and public key encryption
11. Database gatekeeper – controls access to specific databases based on user authentication; useful for developing against multiple databases
12. Protected file system on server – prevents unauthenticated access and snooping of files
13. Logging

Performance

1. Database Connection Pool
2. Object Cache, for sharing system level resources between users
3. Image Cache, implementation for reusing images such as graphs
4. Indexed Data, for quicker data searches

User Interface

1. Page layouts – templates for defining where global elements should be placed and where the content is placed; keeping the content separate from the layout
2. Page skins – settings to define how page elements are displayed, typically with style sheets; colors, borders, images, sizes, etc.
3. Record paging – server-side and client-side elements for navigating a result set
4. Calendar – server-side and client-side elements for displaying a calendar with optional settings; small or large, with icons for holidays and custom events
5. File transfer – server-side and client-side elements for allowing clients to upload, download, and stream files based on permissions and file content-type
6. Graphs – server-side image generation including web browser tool-tips; bar, line, pie, plot, etc.
7. Menu system
    - Top-level menu displaying modules; graphically rich using style sheets
    - Sub-menus for each module; graphically rich using style sheets
    - Tab-style container menus; graphically rich using style sheets
8. Global items – elements that traverse the whole site or a sub-section of page layouts
    - My Items; a list of alerts specific to the user
    - Recent Items; a list of recently accessed items
    - Search; a text field for searching the whole site for related content
    - Quick Actions; a menu for popping up an action item form
    - Help; page-sensitive help
    - QA Development Tool; page-sensitive bug, feature, and help entry
9. Site preferences – allows for installations to easily modify the all terminology
10. Lookup lists – a standard way of using combo-boxes with advanced features for displaying entries that are no longer valid in the list and for allowing items to be modified by an administrator
11. Dynamic Forms – forms expressed as xml and translated by a JSP for a consistent look
12. Taglibs – html tags that allow cached access to users, lookup lists, and various custom html elements
13. Time Zones – every element is sensitive to the user's timezone, both when displaying and entering dates and times

Rules Engine

1. Object events – action triggers a customizable workflow process when an object is inserted, updated, deleted, or selected

2. Scheduled events – a timer triggers a customizable workflow proces
3. Notification manager – responsible for sending and logging messages

System Scheduler

1. Can execute tasks written in Java
2. Can execute native executables

Data Transfer and Exchange

1. HTTP-based XML API – multiple clients can add, update, delete or retrieve records on the server
2. Synchronization through XML API – multiple clients can synchronize tables with the server, the server can also map legacy primary keys to server primary keys on-the-fly
3. Process log through XML API – a timestamped listing of external and internal processes
4. Reader/Writer – a generalized application that can read data from various inputs and write the data to various outputs
5. Direct database connectivity

Document/Data Indexing and Search

1. Class indexers to define object metadata to be saved and indexed
2. Context-sensitive search results with word highlighting

Utilities

1. AppUtils for loading and saving persistent application settings
2. ContentUtils for parsing content and metadata from documents
3. DatabaseUtils for working with databases independent of the database type
4. DateUtils for common date methods
5. FileUtils for common file methods; copying files
6. HTMLUtils for working with HTML elements and data
7. HTTPUtils for remote HTTP and HTTPS connectivity
8. ImageUtils for thumbnails and image conversion
9. JasperReportUtils for compiling reports and generating PDFs

10. ObjectUtils for reflection and serialization
11. SearchUtils for working with result sets
12. SMTPMessage for text and HTML email messages, optionally with attachments
13. StringUtils for common string methods
14. SVGUtils for complex SVG methods
15. XMLUtils for common xml methods; parsing, searching nodes, converting to text
16. ZipUtils for adding files to .zip files

Modules

ConcourseSuite comes with several modules that can be configured. From a developer's perspective, ConcourseSuite modules make use of several framework features, including:

1. User session created by the Login process which defines the user's id, role, and permissions
2. System configuration values that are cached in the servlet context
3. Database Objects that interact with the data stored in the database
4. Document Library framework for uploading and downloading files
5. Folders & Forms for custom fields
6. Import Manager for queueing and processing files for importing into the database
7. Form validation
8. etc.

Modules are configured by the CRM administrator using the Users and Roles web interface. The administrator can decide which user roles have access to the modules and the modules can be turned off completely. ConcourseSuite currently has the following modules:

- My Home Page
- Leads
- Contacts
- Pipeline
- Accounts
- Products
- Quotes
- Orders
- Marketing
- Revenue
- Projects
- Help Desk
- Documents
- Employees
- Reports
- Admin

Help Desk Module

The Help Desk module is a customer support system comprising of the following tightly integrated components:

- Ticket Management
- Knowledge Base Management
- Defect Tracking

Ticket Management

When a ticket is entered, there are several optional relationships, which include Contacts, Service Contracts, Assets, Labor Categories, Documents, Tasks, Action Plans, and Defects. The ticket structure is also reused in Project Management.

Java Classes

Base objects are located at:

org.aspcfs.modules.troubletickets.base.*;

The important base classs used in Ticket Management are Ticket and TicketList

Module actions are located at:

org.aspcfs.modules.troubletickets.actions.*;

The important action class used in Ticket Management is TroubleTickets

Workflow engine components are located at:

org.aspcfs.modules.troubletickets.components.*;

Supporting jasper reports code is located at:

org.aspcfs.modules.troubletickets.jasperreports.*;

JSPs

Help Desk JSPs are located at src/web/troubletickets/* and include several reusable details and form pages used by other modules.

Most of the jsp files related to Ticket Management start with the prefix troubletickets_'.

For example, the ticket details page has the file name troubletickets_details.jsp

Special files with the suffix _include.jsp are reused in other jsp files to improve the code maintenance time.

For example, the file troubletickets_actionplan_work_details_include.jsp is reused in the following modules

1. Help Desk - troubletickets_actionplan_work_details.jsp
2. Accounts - accounts_action_plan_work_details.jsp and accounts_tickets_actionplan_work_details.jsp
3. My Homepage - action_plan_work_details.jsp

Database Tables

ticket

1. org_id – the organization that requested this ticket
2. contact_id – the contact that requested this ticket
3. problem – the issue that was raised in response to creating this ticket
4. closed – the date/time when this ticket was closed, whether solved or not
5. pri_code – the assigned priority
6. level_code (unused)
7. department_code – the department to which this ticket is assigned
8. source_code – the method in which this ticket was submitted: email, phone call, etc.
9. cat_code – the top level category that this ticket maps to
10. subcat_code1 – the category narrowed another level
11. subcat_code2 – the category narrowed another level
12. subcat_code3 – the category narrowed another level
13. assigned_to – the user that is assigned this ticket
14. comment (unused)
15. solution – the solution that was used to solve the issue
16. scode – The severity in which the issue is disruptive of service
17. critical (unused)

ticketlog

When a ticket is updated, the data is analyzed and all changes are appended to a log, including comments.

For example, the ticket workflow might include the following steps, at each step a log record is created:

1. The ticket issue is entered into the database
2. The ticket is assigned to a user
3. The user reviews the ticket and adds a comment

4. The user closes the ticket

These fields resemble a ticket and map back to the user that made the changes.

Workflow Events

Ticket insertion and updation is linked to a specific set of background processes or workflow. The application workflow handles insertion and updation of the contact and account history. The user workflow handles custom user notifications. The currrent workflow also includes a scheduled workflow for Ticket Management to notify users of the presence of unassigned tickets in the Help Desk module.

Object Validation

The object validator for Tickets checks for the presence of the following fields.

1. orgId
2. contactId
3. problem

It also checks for valid input in the case of related objects.

Object validation is performed by the following lines of code in the action class TroubleTickets.java.

```
// set the valid flag to false.
boolean isValid = false;

// validate the ticket object and return "false" to indicate the presence of validation errors.
isValid = this.validateObject(context, db, thisTicket);
```

If the value of isValid is false, the ObjectValidator has added the errors to the ActionContext and the ticket should not be inserted or updated.

The control needs to be returned back to the Add/Modify page for displaying the errors or warnings.

The Add/Modify JSP pages contain the following additional code at the begining of the file to display the generic warning in case of an error in the input fields.

```
<dhv:formMessage />
```

or

```
<dhv:formMessage showSpace="true"/>
```

or

```
<%= showError(request, "actionError") %>
```

The specific error in the field is displayed by adding the following line of code after the specific field.

```
<%= showAttribute(request,"orgIdError") %>
```

Mandatory items have the following html code at the end of the input field in the jsp.

```
<font color="red">*</font>
```

Recent Items

On inserting, modifying or deleting tickets, the recent items have to be updated in the action class as follows:

```
addRecentItem(context, thisTicket); // on viewing or modifying the ticket.
```

```
deleteRecentItem(context, thisTicket); // on deleting the ticket.
```

Knowledge Base Management

Knowledge Base Management involves the assimilation of data related to Tickets based on the categorization of tickets.

A link is provided in the ticket details page to display the relevant knowledge base entries for the selected ticket categories.

Java Classes

Base objects are:

org.aspcfs.modules.troubletickets.base.KnowledgeBase;

org.aspcfs.modules.troubletickets.base.KnowledgeBaseList;

Module action is:

org.aspcfs.modules.troubletickets.actions.KnowledgeBaseManager;

JSPs

Knowledge Base Management JSPs are located at src/web/troubletickets/*.

All the JSP files related to the Knowledge Base Management have a prefix "troubletickets_knowledge_base_".

Database Tables

table knowledge_base

1. kb_id - primary key.
2. category_id - used to store the ticket category related to the knowledge base entry.
3. title - the title of the entry.
4. description - text description of the entry.
5. item_id - related file is stored in the file library and its entry in the table project_files.

Object Validation

Knowledge Base items can not exist without the relevant ticket categorization. Hence the object validator checks for the presence of the related categoryId and the title fields. Though the file attachment is optional for entering a knowledge base entry, an invalid file name will cause the object validation errors.

Defect Tracking

Defect Tracking helps users to track tickets related to a common defect.

Java Classes

Base objects are:

org.aspcfs.modules.troubletickets.base.TicketDefect;

org.aspcfs.modules.troubletickets.base.TicketDefectList;

Module action is:

org.aspcfs.modules.troubletickets.actions.TroubleTicketDefects;

JSPs

Defect Tracking JSPs are located at src/web/troubletickets/*.

All the JSP files related to the Defect Tracking have the prefix "troubletickets_defect_".

The troubletickets_details.jsp is reused to display the ticket details from the defect details page. The attribute defectCheck is used to determine the correct trails in the ticket details page.

Database Tables

table ticket_defect

1. defect_id - primary key
2. title - a concise title of the defect
3. description - text describing the defect in detail
4. start_date - estimated start date of the defect.
5. end_date - estimated date then the defect expires.
6. enabled - used to disable the defect without effecting the start and expiration dates.
7. trashed_date - this date/time field is set to let the application's background trashing process delete the defect and related ticket mapping.
8. site_id - the site selection of the defect.

Object Validation

ObjectValidator checks for the presence of the fields title and startDate on Adding/Modifying TicketDefects.

Implementation Guide

Welcome to ConcourseSuite!

The installation, configuration, maintenance and upgrade of ConcourseSuite is intended to be as simple as possible. However, implementing ConcourseSuite, like any enterprise-class software, is often a significant task for an organization. Implementation requires management participation, technical staff, and often the help of experienced ConcourseSuite consultants.

This document is written for the team of people responsible for implementing ConcourseSuite. The document is broken into roles and phases. In smaller organizations, several roles may be filled by a single individual, while in larger organizations each role may be divided among several people.

This document assumes that organizations implementing ConcourseSuite may use the services of a ConcourseSuite Solution Provider.

Features

The Enterprise and Open-Source editions have the exact same feature-set. The Enterprise edition is pre-compiled, includes easy upgrade scripts, and requires a maintenance contract to foster innovation. The Open-Source edition must be compiled and aggressively maintained.

Implementation Roles

Role	Description
Business Management Role	Responsible for mapping business processes and terminology to ConcourseSuite. The business manager analyzes the organization, constructs flow-charts, determines default ConcourseSuite values and outlines customizations needed by the organization.
Installation Role	Responsible for installing and configuring operating systems, servers, hardware and related CRM software. Additionally responsible for maintenance and disaster recovery plans.
Configuration and Customization Role	Responsible for configuring and setting up ConcourseSuite based on the needs of management. The customizer is also responsible for implementing customizations.

Business Management Role

The business management role oversees the CRM implementation. This role conveys CRM requirements to the other roles.

This role should:

1. Become familiar with the data configuration of CRM
2. Map your organization's business processes and terminology to CRM
3. Define roles in CRM that clearly represent roles in your organization
4. Determine the users of the system

Data Configuration

Roles

Roles allow the administrator to manage groups of permissions granted to users. Changing permissions occurs in real-time, such that adjustments are immediate upon saving the role permissions, even if the user is already logged in. Permissions include having access to modules, and having VIEW/ADD/EDIT/DELETE permissions on module data.

Lookup Lists

All of the drop-down menus are customizable. When lists are modified over time, the values are never permanently deleted. So if there are records using a value, and that value is deleted from the drop-down, existing records will continue to show that value, but with an "*" to indicate the value is no longer current. New records will stop displaying deleted values.

Custom Folders and Fields

This capability allows for custom data to be recorded against records. Not only can you add additional fields to records, but you can add an unlimited set of custom fields to a record.

For example, in the Accounts module you might want to log the organization's Affiliate Id (a custom field), or you might want to record monthly invoices imported from another system (multiple input of custom fields).

Categories

Some capabilities in the CRM allow for hierarchical related data. In the Tickets module, for example, tickets can be categorized up to four (4) levels -- based on the selection in the first category, the second category dynamically adjusts.

Territories

Choose whether or not the "Territories" capability is right for your organization.

The introduction of territories (formerly called Sites) arose out of the corporate need to isolate groups of users and their data. The purpose of territories is to create "silos" of information among users. Users with a territory can only see data within that territory. Top-level users (those without a territory) can see data among all of the territories. These top-level users can participate within each territory and can make assignments to other territory-specific users and non-territory users. As users work within a territory, they are creating referential data which creates dependencies to the territory. Keep in mind that moving data from one territory to another is currently not possible.

Workflow Planning

Action Plans & Workflow

Rules Engine & Events

Installation Role

This role reviews the installation requirements, configures the hardware and software, and performs post-installation tasks.

Configuration and Customization Role

This role is responsible for configuring and setting up the CRM based on the needs of management. The customizer is also responsible for implementing customizations requested by the business management role.

Configuration

- Users
- Roles
- Permissions
- Lookup Lists
- Action Plans
- Fields and Folders

Customization

- Customizing the Look & Feel
- Rules Engine Customization
    - Processes Library
    - Components Library
- Writing CRM Reports
- Portlets

Customizing the Look & Feel

The Centric CRM framework UI is displayed by pulling together a Layout and a Style Sheet, along with some application preferences.

The default look and feel includes the Centric CRM logo, the modules listed across the top, the global items displayed on the right, and the body of the selected page included in the middle. The default also includes a professional color scheme and the "Centric CRM" application title. These elements can easily be modified.

Layout

Style Sheet

Dictionary

Rules Engine Customization

Collaboration in Centric CRM is typically achieved through Action Plans, Assignments, Ownership and Workflows.

Every time a user inserts, updates, or deletes data in Centric CRM, a component-based multi-threaded rules engine is triggered. Events are handled in the background exposing the object before and after it was modified. This can be used for creating an account history item for the selected object, executing a workflow, indexing data, and more, allowing complete customization of an Organization's business processes.

Example Workflow

The following default process occurs every time a ticket is inserted or updated in Centric CRM. The workflow is designed using Centric CRM Action Components and Condition Components. These components are included with Centric CRM. Developers can completely modify this workflow, and others, by changing rules, properties and adding custom components to do just about anything.

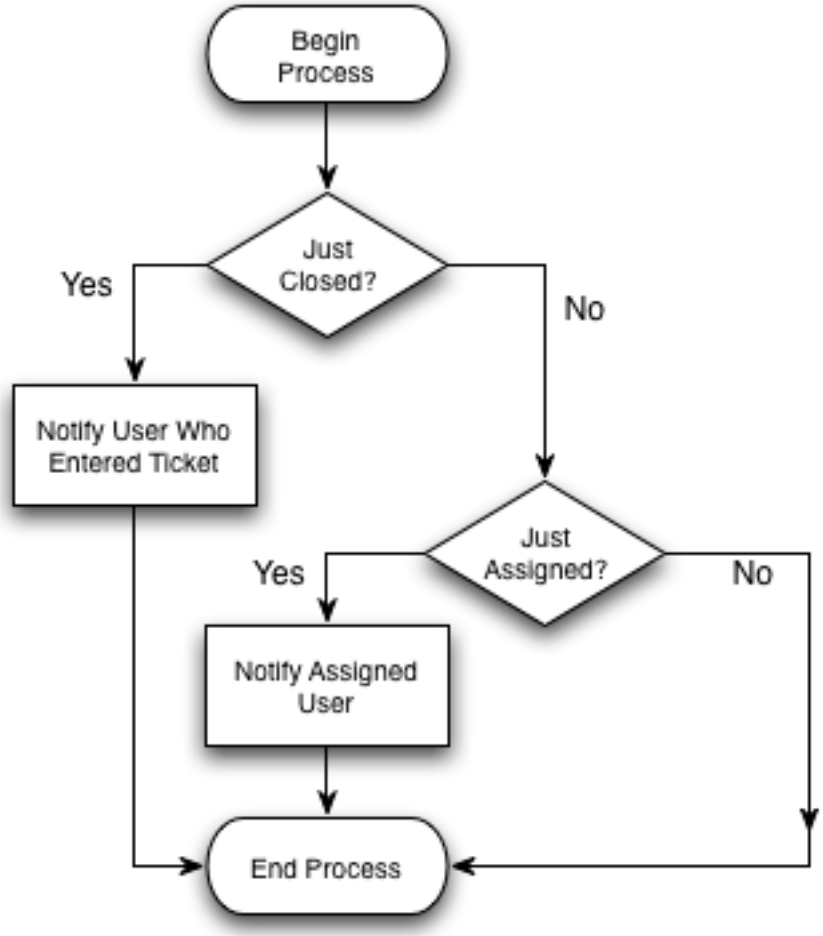
See Creating Workflows to understand the terminology and construction of a workflow and its components. When designing a workflow, it's often best to start with a diagramming tool and create visual flowcharts of the Organization's processes.

Deploying a workflow

Deploying a Workflow using existing Workflow Components:

1. Provide the workflow.xml file to a Centric CRM administrator
2. The administrator navigates to the Admin module, then chooses Configure System, then chooses Business Process Management
3. The administration then uploads the new workflow, optionally replacing any existing workflows -- the default will append the workflows
4. A success message indicates the processes were added

Deploying a Workflow using new components:



1. Package up the custom workflow components into a .jar
2. Copy the .jar into Centric CRM's fileLibrary for backup (a future update will automate deploying these)
3. Copy the .jar into Centric CRM's webapp/WEB-INF/lib directory
4. Restart the Web Application Server
5. Register the custom components with the Centric CRM database (to be documented)
6. Provide the workflow.xml file to a Centric CRM administrator
7. The administrator navigates to the Admin module, then chooses Configure System, then chooses Business Process Management
8. The administration then uploads the new workflow, optionally replacing any existing workflows -- the default will append the workflows
9. A success message indicates the processes were added

Creating Workflows

The ConcourseSuite CRM framework includes a simple, component-based rules engine that can be used asynchronously for object events (events triggered by inserting, updating, or deleting objects) or for events that occur at a scheduled point in time.

A workflow process is comprised of Java components that act as Conditions or Actions. Conditional components usually inspect an object, then decide if the result is true or false. Action components perform an action based on the object, like sending an email.

The business process workflow is defined using XML, which can be imported into a system using the ConcourseSuite CRM Admin Module. The processes get cached when the application starts up and wait until triggered.

Example Workflow

The following process occurs every time a ticket is inserted or updated in ConcourseSuite CRM. The workflow is designed using Action Components and Condition Components. These components are included with ConcourseSuite CRM. Developers can completely modify this workflow, and others, by changing rules, properties and adding custom components to do just about anything.

Workflow XML

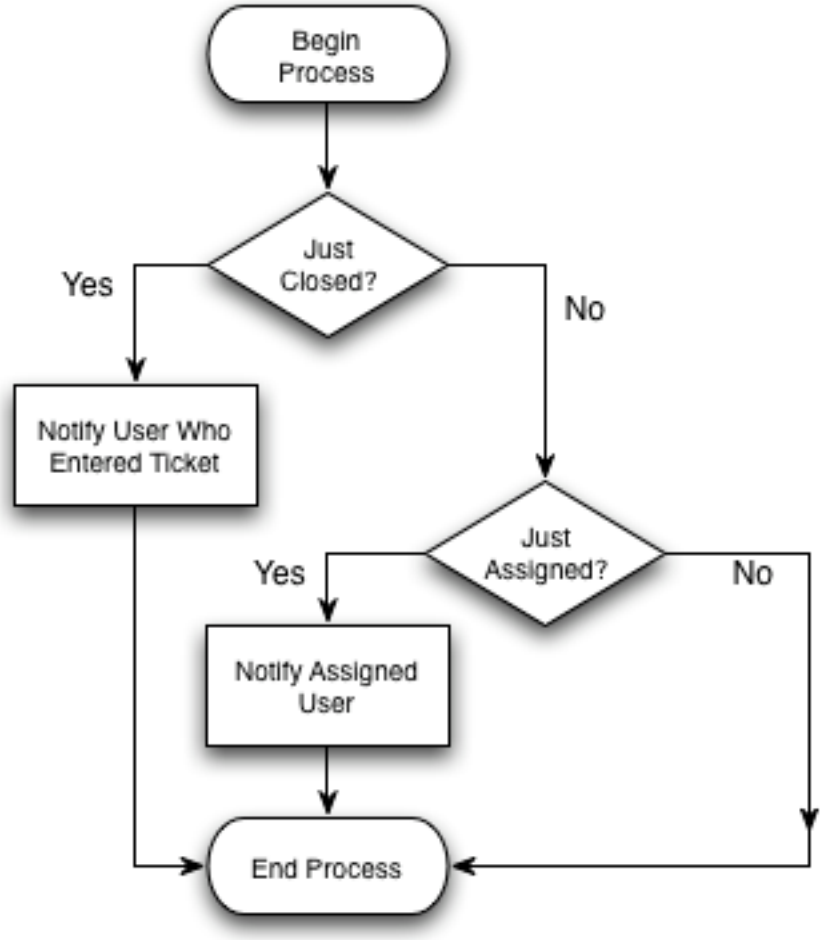
Each process is defined using XML. In the base CRM installation, a workflow_en_US.xml is provided, along with additional translated versions.

The workflow.xml file contains the following information:

- Workflow Processes
- The Objects and Actions which trigger a process
- The Schedules which trigger a process

Defining a Workflow Process

A process has a unique **name,** a **description,** a **startId** which determines the first component in the process, a process **type,** and a **module** id which relates the process to a ConcourseSuite CRM module for viewing online.



The following process starts with component "2" which uses the QueryTicketJustClosed component to determine if the Ticket being inserted or updated has just been closed. This component will return a "true" or "false" and any components that respond to the condition will be triggered accordingly.

```
<processes>
  <process name="dhv.ticket.insert" description="Ticket change notification"
startId="2"
      type="OBJECT_EVENT" module="8">
    <components>
      <component id="2"
class="org.aspcfs.modules.troubletickets.components.QueryTicketJustClosed"/>
      <component id="3" parent="2" if="false"

class="org.aspcfs.modules.troubletickets.components.QueryTicketJustAssigned"/>
      <component id="4" parent="2" if="true"
          class="org.aspcfs.modules.components.SendEmailNotification">
        <parameters>
          <parameter name="notification.module" value="Tickets"/>
          <parameter name="notification.itemId" value="${this.id}"/>
          <parameter name="notification.itemModified" value="${this.modified}"/>
          <parameter name="notification.userToNotify"
value="${previous.enteredBy}"/>
          <parameter name="notification.userGroupToNotify"
value="${previous.userGroupId}"/>
          <parameter name="notification.subject">Ticket Closed:
${this.paddedTicketId}</parameter>
          <parameter name="notification.body"><![CDATA[<strong>The following ticket
in ConcourseSuite CRM
has been closed:</strong><br />
--- Ticket Details ---
<br />
<strong>Ticket #</strong> ${this.paddedTicketId}<br />
Priority: ${ticketPriorityLookup.description}<br />
Severity: ${ticketSeverityLookup.description}<br />
Issue: ${this.problem}<br />
<br />
Comment: ${this.comment}<br />
<br />
Closed by: ${ticketModifiedByContact.nameFirstLast}<br />
<br />
Solution: ${this.solution}<br />
]]></parameter>
        </parameters>
      </component>
      <component id="5" parent="3" if="true"
          class="org.aspcfs.modules.components.SendEmailNotification">
        <parameters>
          <parameter name="notification.module" value="Tickets"/>
          <parameter name="notification.itemId" value="${this.id}"/>
          <parameter name="notification.itemModified" value="${this.modified}"/>
          <parameter name="notification.userToNotify" value="${this.assignedTo}"/>
          <parameter name="notification.subject">Ticket Assigned:
${this.paddedTicketId}</parameter>
          <parameter name="notification.body"><![CDATA[<strong>The following ticket
in ConcourseSuite CRM
has been assigned to you:</strong><br />
<br />
--- Ticket Details ---<br />
<br />
<strong>Ticket #</strong> ${this.paddedTicketId}<br />
Priority: ${ticketPriorityLookup.description}<br />
Severity: ${ticketSeverityLookup.description}<br />
Issue: ${this.problem}<br />
<br />
Assigned By: ${ticketModifiedByContact.nameFirstLast}<br />
Comment: ${this.comment}<br />
]]></parameter>
        </parameters>
```

```
]]></parameter>
        </parameters>
      </component>
      <component id="7" parent="5" if="true"
          class="org.aspcfs.modules.components.SendEmailNotification">
        <parameters>
          <parameter name="notification.module" value="Tickets"/>
          <parameter name="notification.itemId" value="${this.id}"/>
          <parameter name="notification.itemModified" value="${this.modified}"/>
          <parameter name="notification.userGroupToNotify"
value="${this.userGroupId}"/>
          <parameter name="notification.skipUsers" value="${this.assignedTo}"/>
          <parameter name="notification.subject">Ticket Assigned:
${this.paddedTicketId}</parameter>
          <parameter name="notification.body"><![CDATA[<strong>The following ticket
in ConcourseSuite CRM
has been assigned to: ${ticketAssignedToContact.nameFirstLast}</strong><br />
<br />
--- Ticket Details ---<br />
<br />
<strong>Ticket #</strong> ${this.paddedTicketId}<br />
Priority: ${ticketPriorityLookup.description}<br />
Severity: ${ticketSeverityLookup.description}<br />
Issue: ${this.problem}<br />
<br />
Assigned By: ${ticketModifiedByContact.nameFirstLast}<br />
Comment: ${this.comment}<br />
]]></parameter>
        </parameters>
      </component>
      <component id="6" parent="2" if="true"
        class="org.aspcfs.modules.troubletickets.components.SendTicketSurvey"
enabled="false"/>
    </components>
  </process>
</processes>
```

Triggering an Object Based Process

Most ConcourseSuite CRM Module Action Classes call processInsertHook(), processUpdateHook(), and processDeleteHook() when their objects are modified.

The following XML will enable ticket record triggers, when a ticket is inserted or updated in any module, including the HTTP-XML API... the same process is used for insert and update in this example.

```
<hooks>
  <hook class="org.aspcfs.modules.troubletickets.base.Ticket" module="8">
    <actions>
      <action type="update" process="dhv.ticket.insert" enabled="true"/>
      <action type="insert" process="dhv.ticket.insert" enabled="true"/>
    </actions>
  </hook>
</hooks>
```

Triggering a Schedule Based Process

Scheduled processes are great when integrating with other systems or for batch notification.

To schedule a process using standard CRON terminology, the following XML will trigger a process that emails a manager when tickets have not been assigned within 10 minutes.

```
<schedules>
  <schedule>
    <events>
      <event process="dhv.report.ticketList.overdue"
        second="0" minute="*/10" hour="8-18" dayOfMonth="*" month="*" dayOfWeek="*"
year="*"
        extraInfo="" businessDays="true" enabled="true"/>
    </events>
  </schedule>
</schedules>
```

Workflow Components

ConcourseSuite CRM workflow components are Java objects intended to inspect data or act on data. As of Version 4.0, ConcourseSuite CRM ships with 42 "Query" workflow components and 5 "Action" components.

Most components extend the **com.concursive.crm.workflow.ObjectHookComponent** Class which provides easy access to database connections within a component, plus other useful methods. Components also implement the **com.concursive.crm.workflow.ComponentInterface.**

A Simple Component

The following component checks to see if a ticket was reassigned by the user... the result for components is always boolean.

```
public class QueryTicketJustAssigned extends ObjectHookComponent implements
ComponentInterface {

  public String getDescription() {
    return "Was the ticket just assigned or reassigned?";
  }

  public boolean execute(ComponentContext context) {
    Ticket thisTicket = (Ticket) context.getThisObject();
    Ticket previousTicket = (Ticket) context.getPreviousObject();
    if (thisTicket != null) {
      if (previousTicket != null) {
        //Ticket was updated
        return ((thisTicket.getAssignedTo() != previousTicket.getAssignedTo())
            && thisTicket.getAssignedTo() > 0);
      } else {
        //Ticket was inserted
        return (thisTicket.getAssignedTo() > 0);
      }
    }
    return false;
  }

}
```

Registering a Component

ConcourseSuite CRM maintains a list of components in a library, which is read into memory when the web application starts.

Code Action Classes

A module Action Class contains the server-side code that gets executed when the user selects a URL in their browser.

For example, when the user clicks “Generate a Report” the server maps the user request to Java code that prepares the data for displaying back to the user. So, the action might perform database queries, retrieve information from a cache, create Java Beans, etc. When the action is done, all of the data is forwarded to a JSP for returning HTML back to the user's browser.

The Simplest Action Class

The following code is actually quite powerful, although no work is actually being done here. Based on additional ConcourseSuite CRM configuration parameters, the following URL might be enabled...

```
http://127.0.0.1/crm/Prototype.do?command=Default
```

This causes the following code to be executed by the server...

```
package com.concursive.crm.web.modules.prototype.actions;

import com.concursive.commons.web.mvc.actions.ActionContext;
import com.concursive.crm.web.controller.actions.CRMModule;

public final class Prototype extends CRMModule {
public String executeCommandDefault(ActionContext context) {
      return ("DefaultOK");
  }
}
```

Before any code is executed, the Controller will ensure that the browser has already logged into ConcourseSuite CRM before continuing.

Next, if the CRM configuration has a mapping between **Prototype** and **DefaultOK** then the Controller will execute the Default Action of the Prototype Action Class, then based on the return string, forward the response to the mapped action, which might be defined as a JSP or a chained Action.

Important Notes About Action Classes

1. Action Class methods must be in the form **executeCommandName(ActionContext context)** where [name] is the name of an action, like Add; when a request is made that does not specify a command, then executeCommandDefault(ActionContext context) is used
2. **Trap all errors,** with **catch (Exception)** in the action's method and return an OK or Error result to the Controller
3. Always use **finally** to release any resources used by the action, like a database connection, whether the method was successful or not
4. Use a single database connection sparingly and quickly, be sure to free the connection when finished; **getConnection(context)** retrieves a connection for the current user from the pool, and **freeConnection(context, db)** returns it back to the pool; Do not use connection.close() or it will really be closed
5. Store objects in the request to be used by JSPs; **use session objects sparingly** as they require additional memory storage
6. When a POST is finished, use a redirect 302 instead of displaying the page within the POST request.

Extending com.concursive.crm.web.controller.actions.CRMModule

Extending the CRMModule class provides many useful methods that can be used during the action code execution.

Important Methods

- hasPermission() -- to verify user permissions
- getConnection(), freeConnection(), renewConnection() -- to retrieve and reuse database connections
- getUserId() -- to get the logged in user's id
- getUserRange() -- to get the user ids of the logged in user and the ids of those in that user's hierarchy as CSV
- getUserSiteId() -- to get the logged in user's site id, or -1 if none
- getUser() -- to get the logged in user's User session
- getPagedListInfo() -- to get the specified pagedListInfo object for a specific record list view

- getUserTimeZone() -- to get the logged in user's time zone
- hasAuthority() -- to verify access to a record
- validateObject() -- to validate a CRM object
- processInsertHook(), processUpdateHook(), and processDeleteHook() -- to activate the workflow engine
- addRecentItem(), deleteRecentItem(), getRecentItem() -- to work with the "recent items" list
- getSystemStatus() -- to retrieve the system preferences that this user belongs to
- getPath(), getDbNamePath(), getDatePath(), getWebInfPath -- returns a path to the file library
- isRecordAccessPermitted(context, db, int) , isRecordAccessPermitted(ActionContext, Object) -- Check record permissions for the user on an object
- isPortalUser() -- returns whether the current user is using the portal interface or not
- indexAddItem(), indexDeleteItem() -- to add or remove objects from the search index
- executeJob() -- to force execution of scheduled or unscheduled jobs
- getViewpointInfo() -- to get details about the user's current viewpoint

Debugging

ConcourseSuite CRM has been configured to use Apache Commons Logging.

If the web application was installed with debugging turned on, then debug output will be sent to a log file in ${TOMCAT_HOME}/logs/; on Linux, Mac, and Sun the file is catalina.out and on Windows the files are stdout.txt and stderror.txt.

In any class the following code convention is used to output debug info:

```
LOG.debug("Some text");
```

Code with System.out messages will not be accepted. System.out messages can slow down a production web server when enabled and use a lot of disk space.

hasPermission()

Every action in ConcourseSuite CRM requires a user permission check. Based on the user's role, various permissions are enabled and disabled.

Use **hasPermission(context, "permission name-attribute")** to verify that the user has the specified permission before continuing execution of the current action.

- Permissions are generally in the form of "module-feature-action" and are defined in the permissions.xml file.
- The permission check is usually performed first in an Action Class, however multiple checks can be made
- Users should never see an error message that they do not have access to perform the function; the HTML view should hide those things that the user does not have access to

The following Action Class checks to see if the user has access to perform this action...

```
public String executeCommandSearchForm(ActionContext context) {
  if (!hasPermission(context, "sales-view")) {
    return ("PermissionError");
  }
  SystemStatus systemStatus = this.getSystemStatus(context);
  Connection db = null;
  try {
    ...
  }
}
```

The permission is named "sales" and the attribute is "view" -- as described in Create Install and Upgrade Scripts under the Permission Entry section.

Create Install and Upgrade Scripts

When installing ConcourseSuite CRM for the first time, a series of SQL statements, Java Applications, and BeanShell Scripts are executed.

- The SQL statements create database tables and referential integrity
- A Java application installs default data into the tables, based on Locale, from permissions_en_US.xml
- A Java application installs default lookup data into the tables, based on Locale, from lookuplists_en_US.xml
- A Java application installs default help data into the tables, based on Locale, from help.xml

Updating the Install Scripts

SQL Statements

In the ConcourseSuite CRM code structure, there are installation directories for each database type. In each database specific subdirectory, there are files in the form of "new_*.sql" -- these files are read in a specific order by build.xml during database installation.

If possible, edit the existing new_*.sql file with your changes or additions if applicable. If your additions are unrelated to the existing files, then create a new_*.sql file and register it in build.xml.

For your additions and changes to work under different database platforms, database specific files are provided and should also be updated. At some point these are synchronized by the core team.

Permissions XML

The permissions.xml file is read in during database installation and includes:

- Module Names
- Module Permissions
- Module Editors
- Module Reports
- User Roles and default permissions

If making a change to an existing module, be sure to update the capabilities, permissions, and related capability details for that module. If adding a new module, model the new module after an existing module.

A snippet from the Help Desk module appears as:

```
<category id="8" name="Help Desk" folders="true" lookups="true" reports="true"
    scheduledEvents="true" objectEvents="true"
    categories="true" actionPlans="true" customListViews="true">
  <folder constantId="11072003" description="Tickets"/>
  <lookup constantId="922051718" class="lookupList" table="lookup_ticket_cause"
    description="Ticket Cause"/>
  <lookup constantId="922051719" class="lookupList"
table="lookup_ticket_resolution"
    description="Ticket Resolution"/>
  <permission name="tickets" attributes="v---" description="Access to Help Desk
module"/>
  <permission name="tickets-tickets" attributes="vaed" description="Ticket
Records"/>
  <permission name="tickets-tickets-tasks" attributes="vaed" description="Tasks"/>
  <permission name="tickets-knowledge-base" attributes="vaed"
description="Knowledge Base"/>
  <permission name="tickets-defects" attributes="vaed" description="Ticket
Defects"/>
  <report file="tickets_department.xml" type="user" permission="tickets-tickets"
    title="Tickets by Department"
    description="What tickets are there in each department?"/>
  <report file="ticket_summary_date.xml" type="user" permission="tickets-tickets"
    title="Ticket counts by Department"
    description="How many tickets are there in the system on a particular date?"/>
  <report file="assets_under_contract_report.xml" type="user" permission="tickets-
tickets"
    title="Assets Under Contract"
    description="Which assets are covered by contracts?"/>
  <multipleCategory constantId="202041401" table="ticket_category" maxLevels="4"
    description="Ticket Categories"/>
  <actionPlanEditor constantId="912051329" description="Action Plans related to
Tickets"/>
  <customListViewCategory constantId="113051436" description="Ticket Custom List
Views"/>
</category>
```

The Help Desk module is declared with a unique constant **id,** a **name,** and the following editor capabilities:

1. Folders -- enables the Administrator module editor for modifying Help Desk folders
2. Lookups -- enables the Administrator module editor for modifying Help Desk lookup lists
3. Reports -- enables Help Desk reports in the Reports module
4. ScheduledEvents -- enables displaying Help Desk scheduled events in the Administrator module

5. ObjectEvents -- enables displaying Help Desk object events in the Administrator module
6. Categories -- enables the Administrator module for modifying Help Desk categories
7. ActionPlans -- enables the Administrator module for configuring Help Desk action plans
8. CustomListViews -- enables the Administrator module for configuring Help Desk list views

For each capability, a corresponding entry must be created.

Folder Entry

Folders have a unique **constantId** and a **description.**

```
<folder constantId="11072003" description="Tickets"/>
```

Lookup List Entry

Lookup lists have a unique **constantId,** specify a **class** which defines how the lookup list is edited, the **table** to be modified, and a **description.** The following entry will register a lookup list against a specific module. The default data for a lookup list is stored in the Lookup List XML.

```
<lookup constantId="922051719" class="lookupList" table="lookup_ticket_resolution"
  description="Ticket Resolution"/>
```

Permission Entry

Permissions have a unique **name, description,** and all of the **attributes** that have been implemented by the developer

Attributes map to the web-based role editor as:

- **v** Access to the specified module or module area; or Access to view records of the specified type
- **a** Access to add records of the specified type
- **e** Access to edit records of the specified type
- **d** Access to delete records of the specified type

```
<permission name="tickets" attributes="v---" description="Access to Help Desk
module"/>
<permission name="tickets-tickets" attributes="vaed" description="Ticket Records"/>
```

Report Entry

Reports are associated with a **file,** have an access **type,** can be mapped to a user **permission,** have a short **title,** and have a lengthier **description.**

```
<report file="assets_under_contract_report.xml" type="user" permission="tickets-
tickets"
  title="Assets Under Contract" description="Which assets are covered by
contracts?"/>
```

Multiple Category Entry

```
<multipleCategory constantId="202041401" table="ticket_category" maxLevels="4"
  description="Ticket Categories"/>
```

Action Plan Editor Entry

```
<actionPlanEditor constantId="912051329" description="Action Plans related to
Tickets"/>
```

Custom List View Category Entry

```
<customListViewCategory constantId="113051436" description="Ticket Custom List
Views"/>
```

Role Entry

Further down in the file there are default User Roles that also get installed. For each role, a **name** is identified, a lengthier **description,** and the permissions and access attributes that this role has.

```
<role name="Sales Manager" description="Manages all accounts and opportunities">
   <permission name="myhomepage" attributes="v---"/>
   <permission name="myhomepage-dashboard" attributes="v---"/>
   <permission name="myhomepage-inbox" attributes="v---"/>
   <permission name="myhomepage-tasks" attributes="vaed"/>
   <permission name="myhomepage-reassign" attributes="v-e-"/>
   <permission name="myhomepage-profile" attributes="v---"/>
   <permission name="myhomepage-profile-personal" attributes="v-e-"/>
   <permission name="myhomepage-profile-settings" attributes="v-e-"/>
   <permission name="myhomepage-profile-password" attributes="--e-"/>
   <permission name="myhomepage-action-lists" attributes="vaed"/>
   <permission name="myhomepage-action-plans" attributes="vaed"/>
   <permission name="sales" attributes="v---"/>
   <permission name="sales-leads" attributes="vaed"/>
   <permission name="sales-import" attributes="v---"/>
</role>
```

Lookup Lists XML

Updating the Upgrade Scripts

Upgrade scripts are also stored in the **src/sql** directory structure and can be either database specific .sql files, or database and language independent .bsh files.

SQL scripts are used for making changes to the schema, BeanShell scripts are used for data related additions and changes.

Since ConcourseSuite CRM is localized and supports multiple database servers, using a .sql script is usually not acceptable for inserting String values into the database. In these cases, a BeanShell script must be created to insert the values.

SQL Upgrade Scripts

BeanShell Upgrade Scripts

Code Structure

The Centric CRM source code is available from the Centric CRM Community web site. Starting with Centric CRM Version 4.0, the source code can be accessed from the Centric CRM Subversion server.

Register with the Centric CRM site, then use your username and password, and a subversion client, to checkout the latest working code.

Subversion Branches Available	
Centric CRM v4.0 Release Branch	https://svn.centricsuite.com/webapp/branches/branch-40
Centric CRM v4.1 Release Branch	https://svn.centricsuite.com/webapp/branches/branch-41
Centric CRM v5.0 Release Branch	https://svn.centricsuite.com/webapp/branches/branch-42

Going forward, all releases will be tagged, branched, and merged back into the TRUNK so that they can be accessed and have fixes applied.

Key Directories and Files

Path	Description
build.xml	Apache Ant build script
master.properties	List of configurable Centric CRM properties -- edit a copy only
build.cleanup	Outdated files that will be deleted from the web server during the build process
src/	All of the application source code is split into subdirectories here
src/java/	Java source code
src/languages/	The language translation files that are exported from the Centric CRM web site
src/sql/	SQL and BeanShell scripts used for installing and upgrading databases
src/web/jsp/	JSP files
src/web/css/	CSS files
src/web/images/	Web optimized images
src/web/WEB-INF/	Web application configuration files
lib/	3rd party libraries
pref/	Configuration and language translation files

build/	Temporary files generated during the build process

Developer Tools

The following are platform-independent tools recommended for working on Centric CRM and framework code and documentation. Developers should use an IDE that they and/or their organization are comfortable in using.

Build Tool

Centric CRM uses Apache Ant for compiling and deployment. Version 1.6.2 or higher is required.

IDE/Editor

Recommended editors include JEdit, Eclipse, JetBrains IntelliJ and NetBeans. These editors work well with web applications and each of the file types you will be working with.

Source Control

If your IDE doesn't have Subversion support, then we recommend using SmartSVN or the command-line tools for Subversion.

For more information, the SVN book is available online.

Reporting Tools

The JasperReports library is included with the Centric CRM source code, but check the JasperReports website for documentation and license information.

iReport is a GUI front-end for JasperReports. The download includes everything needed to design and work with JasperReports documents visually. When using iReport, it is necessary to use a version compatible with the version of JasperReports used in Centric CRM.

getConnection()

The methods **getConnection(), freeConnection(),** and **renewConnection()** all work together to provide the Action Class with interaction with the database connection pool.

The following code shows example usage of **getConnection()** and **freeConnection()** from an Action Class:

```
Connection db = null;
try {
  // Get a database connection from the pool
  db = getConnection(context);
  // Perform lots of work with the connection
} catch (Exception e) {
  context.getRequest().setAttribute("Error", e);
  return ("SystemError");
} finally {
  // Return the database connection to the pool
  this.freeConnection(context, db);
}
```

ConcourseSuite CRM has default properties in "build.properties" that specifies the maximum amount of time that a connection can be retained.

```
CONNECTION_POOL.MAX_DEAD_TIME.SECONDS=300
```

If the connection is not renewed with **renewConnection(db)** then the connection will be assumed to be misused and destroyed after the time limit has expired.

Since some operations in ConcourseSuite CRM may require a lengthy database connection, the connection can be renewed often to prevent recycling.

processInsertHook()

The methods **processInsertHook(), processUpdateHook(),** and **processDeleteHook()** are used to trigger workflow processes.

Trigger Workflow Manager During Object Insert

The following Module Action Class code shows that when an object is inserted, the workflow manager is notified, whether there are workflow processes defined to execute on this object or not.

```
// Retrieve ticket from request
Ticket ticket = (Ticket) context.getFormBean();
// Set server side properties
ticket.setEnteredBy(getUserId(context));
ticket.setModifiedBy(getUserId(context));
boolean isValid = validateObject(context, db, ticket);
if (isValid) {
  boolean recordInserted = ticket.insert(db);
  if (recordInserted) {
    // Trigger the workflow manager
    processInsertHook(context, ticket);
  }
}
```

Trigger Workflow Manager During Object Update

The following Module Action Class code shows that when an object is updated, the workflow manager is notified, whether there are workflow processes defined to execute on this object or not.

```
// Retrieve ticket from request
Ticket ticket = (Ticket) context.getFormBean();
// Set server side properties
ticket.setModifiedBy(getUserId(context));
boolean isValid = validateObject(context, db, ticket);
if (isValid) {
  // Load the previous ticket for comparison
  Ticket previousTicket = new Ticket(db, ticket.getId());
  int resultCount = ticket.update(db);
  if (resultCount == 1) {
    // Get the updated ticket details
    ticket.queryRecord(db, ticket.getId());
    // Trigger the workflow manager
    processUpdateHook(context, previousTicket, ticket);
  }
}
```

Trigger Workflow Manager During Object Deletion

The following Module Action Class code shows that when an object is deleted, the workflow manager is notified, whether there are workflow processes defined to execute on this object or not.

```
// Load the ticket, given a ticket id
Ticket ticket = new Ticket(db, id);
// Attempt to delete the ticket and any associated ticket documents
boolean recordDeleted = ticket.delete(db, getDbNamePath(context));
if (recordDeleted) {
  // Trigger the workflow manager
  processDeleteHook(context, ticket);
}
```

isRecordAccessPermitted(context, db, int)

This is a wrapper method that checks record permissions for a user. It checks

- Permissions on the record for portal users (a capability available to account contacts) as they are restricted to only view or add certain information about the account for which they are a contact.
- Permissions for users who may belong to a division or site. Such users are restricted to access information from their site only. A user who is not assigned a site (i.e., -1) has access data from all sites. This method is used when there exists a relationship between an Organization record(which has a siteId) and the record (which may not have a siteId) for which permissions need to be checked.

```
  protected static boolean isRecordAccessPermitted(ActionContext context,
Connection db, int tmpOrgId) throws SQLException {
    if (isPortalUser(context)) {
      if (tmpOrgId == getPortalUserPermittedOrgId(context)) {
        return true;
      } else {
        return false;
      }
    } else {
      if ((UserUtils.getUserSiteId(context.getRequest())) != -1) {
        int orgSiteId = Organization.getOrganizationSiteId(db, tmpOrgId);
        if (orgSiteId == UserUtils.getUserSiteId(context.getRequest())) {
          return true;
        } else {
          return false;
        }
      } else {
        // has permission to view records of all sites
        return true;
      }
    }
  }
```

isRecordAccessPermitted(ActionContext, Object)

This is a method that checks record permissions for a user. It checks permissions for users who may belong to a division or site. Such users are restricted to access information from their site only. A user who is not assigned a site (i.e., -1) has access data from all sites. This method is used when site information exists in the record for which permissions need to be checked.

```
  protected static boolean isRecordAccessPermitted(ActionContext context, Object
object) throws Exception {
    int tmpUserSiteId = UserUtils.getUserSiteId(context.getRequest());
    if (tmpUserSiteId != -1) {

      Method method = object.getClass().getMethod(
          "getSiteId", (java.lang.Class[]) null);
      Object result = method.invoke(object, (java.lang.Object[]) null);
      int tmpObjectSiteId = ((Integer) result).intValue();

      if (tmpObjectSiteId == tmpUserSiteId) {
        return true;
      } else {
        return false;
      }
    } else {
      // has permission to view records of all sites
      return true;
    }
  }
```

NOTE: This validation uses reflection to examine the getSiteId() method of the object for which record access needs to be checked, hence it is mandatory for these classes to have the getSiteId method.

Build Process

An Ant build script is used for initializing, installing, and upgrading Centric CRM from source. The same script is used on Linux, Mac, Sun and Windows systems.

Ant Targets

Executing "ant" without any parameters will display a list of ant targets.

```
Buildfile: build.xml

usage:
     [echo] ant compile: compile the sources
     [echo] ant test-compile: compile the tests
     [echo] ant test: compile and run the tests using a test database
     [echo] ant test -Dtest=specificTest: compile and run a specific test using a
test database
     [echo] ant package: compile and generate a war
     [echo] ant package-tools: create a distributable tools jar file
     [echo] ant site: generate java docs and web site info
     [echo] ant clean: delete temporary files used in build
     [echo]
     [echo] ant deploy-webapp: deploy the webapp as an exploded directory
     [echo] ant deploy-tomcat: deploy the webapp .war into a running Tomcat
instance
     [echo] ant installdb: install the database from the commandline
     [echo]     install.database: just create the tables
     [echo]     install.help: just install the help contents
     [echo] ant upgradedb: upgrade the database from the commandline
     [echo]
```

Configuration Steps

The build process needs to be configured before ConcourseSuite CRM can be compiled and deployed.

Each time "ant package" is executed, the build process verifies the environment, alerting you to any changes that need to be made.

Configuration Steps:

1. Stop Tomcat

2. Run "ant package" to initialize the configuration paths and files
3. Make changes as requested to the command line environment; then run "ant package" until successful
4. Copy the resulting .war file to Tomcat's web apps path
5. Start Tomcat
6. Using a browser navigate to the deployed CRM for additional configuration options

Build.Properties

After configuration, all properties are updated and stored in the build.properties file of the file library.

Property	Description
SYSTEM.LANGUAGE	Default language setting: even though any locale can be specified, translations and supporting database data needs to be available for this to work
DEBUGLEVEL	If uncommented, then all application debug output is sent to system.out

Advanced Configuration (Optional)

Web-based configuration of ConcourseSuite CRM is recommended and is the default setting. However, this option has an override so that the developer can work with multiple databases based on virtual hosts.

Without overriding, all requests to the CRM use the exact same settings and database. In this case, http://127.0.0.1/crm always uses the same database.

When CONTROL=BYPASS_WEB-BASED_APPLICATION_SETUP is uncommented, ConcourseSuite CRM can use multiple databases based on the virtual host and a mapping in the [sites] table.

Using the HTTP-XML API

Concursive includes a multi-platform accessible XML API using HTTP for communicating with the ConcourseSuite CRM server application by third-party applications. The XML API exposes read and write capabilities to client systems. Essentially, the client posts XML data using a well-formed HTTP 1.1 or 1.0 request to the server and receives a status, as well as any requested records, back in the response.

For example, you can capture leads or tickets from your existing web site and send them straight into the CRM. You can also read data from the CRM that can be used in an external application or web site.

The following sections will provide information on using the HTTP-XML API:

- Server Setup for enabling HTTP-XML API
- XML API Protocol
- XML API Reference
- Tools Package (for Java)
- XML API for PHP
- XML API Examples
- FAQ

Server Setup for enabling HTTP-XML API

There are essentially 2 ways of interacting with the remote CRM Server. The server allows a CRM user to connect to the server remotely and perform data operations on the server. An external client can also connect to ConcourseSuite and perform data operations. Both the external user and client are required to authenticate with the server before being allowed to perform operations.

CRM User Mode

An Administrator can configure a CRM User's access to the XML API by setting the User's [allow_httpapi_access] property to true/false. By default all CRM User's are allowed to access the API. This allows for Offline and Mobile clients to access a user's own CRM data.

Note: When a user is communicating with the XML API, the 'code' property sent as part of the authentication token, should correspond to the user's MD5 Hash of the password. This code will be compared with the user's password in the CRM to validate the user.

Client Mode

Every external client application that needs remote access to ConcourseSuite, should have a corresponding [sync_client] record. *HTTP-XML API Client Manager* allows a CRM Administrator to manage Clients. The admin can add new clients using this interface when required.

The following Client properties are of importance and need to be provided:

1. type = any arbitrary name that describes the client
2. version = any arbitrary name that describes the version of the client
3. enabled = set to 'true' if client needs to be allowed access
4. code = a string or token that the client will use during authentication

XML API Protocol

Both the CRM Server and the Client (Application Client or Mobile User Client) can communicate by sending XML packets. The Client is required to understand the structure of the XML response, so that it can process any records sent by the Server.

XML-HTTP

The Client initiates the data transfer procedure by establishing an HTTP/S connection with the Server. Once the connection is active, the client posts XML data to the Server using HTTP POST. The Server will process the data and return a status along with any data that was requested.

HTTP Packet

The raw HTTP must include the following:

1. Request Method set to "POST" along with http://www.example.com/ProcessPacket.do path
2. The Host specified with the host name that is used to access the CRM installation; like www.example.com
3. Content-Type set to "text/xml"
4. Content-Length set to the number of characters of the XML being submitted
5. commit-level set to either "packet" or "transaction", defaults to 'packet'
6. object-validation set to "true" or "false", default to 'true'
7. Two (2) empty lines
8. The XML string, defined in the next section
9. Optional Cookie information so that additional requests are handled quicker

```
POST http://www.example.com/ProcessPacket.do HTTP/1.0
Host: www.example.com
Content-Type: text/xml
Context-Length: 11


<xml></xml>
```

XML Request Packet

Each time a new HTTP request is made, the Client must include authentication information and one or more transactions in the body content. The encoding method provided by the Client in the request will be used in the reponse. If no encoding is specified, then UTF-16 is used by default.

1. Authentication

The Client identifies itself to the server by sending XML authentication information to the server. The server looks up the client information, and if successful processes the transaction. If authentication fails, a failed message is returned to the client.

The authentication information includes the following items:

- id- The host the client is connecting to
- code- If it is a sync client, then provide [sync_client].code else if user, the MD5(password) Hash
- systemId- A Server assigned id to map transactions to (allows for multiple systems to be interacting)
- clientId- Provide only if it is a Client else don't include
- username- Provide only if it is a CRM User else don't include

2. Transaction

Within the same request as the authentication information, one or more transactions must be specified. The transactions describe to the server an action to be performed. For example, the action could be to insert an object, update an object, or query a list of objects. Depending on the transaction type, meta data can be included to describe the requested information from the server.

Transactions include the following items:

- Meta information to describe data related to the action being performed
- The action to perform

An action is composed of the following items:

1. id- a client supplied transaction id to reference the transaction status responses; the transaction id is valid only for the given request/response and does not affect other

client's requests/responses.

2. objectName- the object on which an action is to be performed
3. action- the action to be performed on the object

3. Example Client XML Request

```
<?xml version="1.0" encoding="UTF-8"?>
<app>
  <authentication>
    <id>www.your-crm-server.com</id>
    <systemId>4</systemId>
    <username>XXX</username>
    < code>** User's MD5 Password< /code>
  </authentication>
  <transaction id="1">
    <meta>
      <property>id</property>
    </meta>
    <account action="insert">
      <name>Dark Horse Ventures</name>
    </account>
  </transaction>
</app>
```

4. XML Response Packet

When the server completes the requested transactions, an XML HTTP response is sent back to the client. The response includes a status corresponding to each transaction requested by the client, and any other data pertaining to the requested action.

1. Status; 0 = Success, Non-Zero = Error
2. ErrorText; if no error then no text, otherwise a message is provided
3. Example XML Response

```
<?xml version="1.0" encoding="UTF-8"?>
<aspcfs>
  <response id="1">
    <status>0</status>
    <errorText/>
    <recordSet name="accountList" count="1">
      <record action="insert">
        <name>Dark Horse Ventures</name>
      </record>
    </recordSet>
  </response>
</aspcfs>
```

XML API Reference

Transactions sent as part of the XML packet are processed by the server and the required action is performed. Each transaction is made up of an action with a corresponding object to perform the action on.

1. XML Actions

The Client can specify one of the following actions to perform on the object:

- **insert:** Inserts the specified object in the database
- **update:** Updates the specified object in the database
- **delete:** Deletes the specified object from the database
- **select:** Queries the specified object type in the database; a list of objects can be returned, as well as a subset based on criteria.

2. XML Objects

Each object has a list of properties that can be set, depending on the action. The mapping table shown below specifies the objects that can be accessed through the XML API. In the mapping table, the id attribute is the name used to access the object. Each object has a list of properties that can be set. Some of the properties(fields) reference other tables.

The attributes of a property(field) include:

1. **name:** The name of the property that can be set

2. **alias:** Typically used with mapping the client's primary key to the server's. If an alias called 'guid' is used, then the client's primary key will be compared to the server's primary key. This is essentially used when a client has to perform synchronization with the server. For more information on the Sync Protocol and Sync API, refer to the Client - Server Synchronization section.

3. **type:** The expected value type of the property

4. **lookup:** Typically used with mapping the client's primary key to the server's; If a lookup attribute is specified then the value will be looked up in the client-server mapping before the operation is performed.

3. XML Object Mapping

The following modules have support for querying objects through the API. Click on a particular module below to see a list of Objects that belong to the module and the properties that can be specified for each object via the API.

XML API: Lookup Lists

XML API: Lookup Lists for Account

The following lookups for Accounts &Contacts objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
lookupAccountTypes	LookupList	-	Y	NT	Y
lookupAccountTypesList	LookupList	Y	-	-	-
lookupAccountSize	LookupList	-	Y	NT	Y
lookupAccountSizeList	LookupList	Y	-	-	-
lookupAccountStage	LookupList	-	Y	NT	Y
lookupAccountStageList	LookupList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A **NT** =Not Tested

LookupList

```
code
description
level
enabled
entered
modified
```

XML API: Lookup Lists for Contacts

The following lookups for Contacts objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
lookupContactAddressTypes	LookupList	-	Y	NT	Y
lookupContactAddressTypes List	LookupList	Y	-	-	-
lookupContactEmailTypes	LookupList	-	Y	NT	Y
lookupContactEmailTypes List	LookupList	Y	-	-	-
lookupContactPhoneTypes	LookupList	-	Y	NT	Y
lookupContactPhoneTypesList	LookupList	Y	-	-	-
lookupContactSource	LookupList	-	Y	NT	Y

lookupContactSource List	LookupList	Y	-	-	-
lookupContactRating	LookupList	-	Y	NT	Y
lookupContactRatingList	LookupList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A **NT** =Not Tested

LookupList

```
code
description
level
enabled
entered
modified
```

XML API: Lookup Lists for Roles, Action Lists,Relations&Reports

The following lookups for Roles, Action Lists,Relations&Reports objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
lookupDepartment	LookupList	-	Y	NT	Y
lookupDepartment List	LookupList	Y	-	-	-
lookupDeliveryOptions	LookupList	-	Y	NT	Y
lookupDeliveryOptionsList	LookupList	Y	-	-	-
lookupEmploymentTypes	LookupList	-	Y	NT	Y
lookupEmploymentTypes List	LookupList	Y	-	-	-
lookupInstantMessenger Types	LookupList	-	Y	NT	Y

lookupInstantMessengerTypes List	LookupList	Y	-	-	-
lookupLocale	LookupList	-	Y	NT	Y
lookupLocaleList	LookupList	Y	-	-	-
lookupOrgAddressTypes	LookupList	-	Y	NT	Y
lookupOrgAddressTypesList	LookupList	Y	-	-	-
lookupOrgEmailTypes	LookupList	-	Y	NT	Y
lookupOrgEmailTypesList	LookupList	Y	-	-	-
lookupOrgPhoneTypes	LookupList	-	Y	NT	Y

lookupOrgPhoneTypesList	LookupList	Y	-	-	-
lookupSegments	LookupList	-	Y	NT	Y
lookupSegmentsList	LookupList	Y	-	-	-
lookupTitle	LookupList	-	Y	NT	Y
lookupTitleList	LookupList	Y	-	-	-
lookupTextMessageTypes	LookupList	-	Y	NT	Y
lookupTextMessageTypesList	LookupList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A **NT** =Not Tested

LookupList

```
code
description
level
enabled
entered
modified
```

XML API: Lookup Lists for Opportunities & Calls

The following Lookups for Opportunities & Calls objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
lookupCallTypes	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupCallTypesList	org.aspcfs.utils.web.LookupList	Y	-	-	-
lookupOpportunityEnvironment	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupOpportunityEnvironment List	org.aspcfs.utils.web.LookupList	Y	-	-	-
lookupOpportunityCompetitors	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupOpportunityCompetitors List	org.aspcfs.utils.web.LookupList	Y	-	-	-

lookupOpportunityEventCompelling	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupOpportunityEventCompellingList	org.aspcfs.utils.web.LookupList	Y	-	-	-
lookupOpportunityBudget	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupOpportunityBudgetList	org.aspcfs.utils.web.LookupList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A **NT** =Not Tested

org.aspcfs.utils.web.LookupList

```
code
description
level
enabled
entered
modified
```

XML API: Lookup Lists for Product Catalogs

The following Lookups for product Catalogs objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
lookupCurrency	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupCurrencyList	org.aspcfs.utils.web.LookupList	Y	-	-	-
lookupProductCategoryType	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupProductCategoryTypeList	org.aspcfs.utils.web.LookupList	Y	-	-	-
lookupProductType	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupProductTypeList	org.aspcfs.utils.web.LookupList	Y	-	-	-

lookupProductManufacturer	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupProductManufacturerList	org.aspcfs.utils.web.LookupList	Y	-	-	-
lookupProductFormat	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupProductFormatList	org.aspcfs.utils.web.LookupList	Y	-	-	-
lookupProductShipping	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupProductShippingList	org.aspcfs.utils.web.LookupList	Y	-	-	-
lookupProductShipTime	org.aspcfs.utils.web.LookupList	-	Y	NT	Y

lookupProductShipTimeList	org.aspcfs.utils.web.LookupList	Y	-	-	-
lookupProductTax	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupProductTaxList	org.aspcfs.utils.web.LookupList	Y	-	-	-
lookupRecurringType	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupRecurringTypeList	org.aspcfs.utils.web.LookupList	Y	-	-	-
lookupProductConfResult	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupProductConfResultList	org.aspcfs.utils.web.LookupList	Y	-	-	-

lookupProductKeyword	org.aspcfs.utils.web.LookupList	-	Y	NT	Y
lookupProductKeywordList	org.aspcfs.utils.web.LookupList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A **NT** =Not Tested

org.aspcfs.utils.web.LookupList

```
code
description
level
enabled
entered
modified
```

XML API: Lookup Lists for Tickets

The following Lookups for Tickets objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
lookupTicketLevel	LookupList	-	Y	NT	Y
lookupTicketLevelList	LookupList	Y	-	-	-
lookupTicketSource	LookupList	-	Y	NT	Y
lookupTicketSourceList	LookupList	Y	-	-	-
lookupTicketStatus	LookupList	-	Y	NT	Y
lookupTicketStatusList	LookupList	Y	-	-	-
lookupTicketEscalation	LookupList	-	Y	NT	Y
lookupTicketEscalationList	LookupList	Y	-	-	-

lookupTicketCause	LookupList	-	Y	NT	Y
lookupTicketCauseList	LookupList	Y	-	-	-
lookupTicketResolution	LookupList	-	Y	NT	Y
lookupTicketResolutionList	LookupList	Y	-	-	-
lookupTicketState	LookupList	-	Y	NT	Y
lookupTicketStateList	LookupList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A **NT** =Not Tested

LookupList

```
code
description
level
enabled
entered
modified
```

XML API: Lookup Lists for Quotes

The following Lookups for Quotes objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
lookupQuoteStatus	LookupList	-	Y	NT	Y
lookupQuoteStatusList	LookupList	Y	-	-	-
lookupQuoteType	LookupList	-	Y	NT	Y
lookupQuoteTypeList	LookupList	Y	-	-	-
lookupQuoteTerms	LookupList	-	Y	NT	Y
lookupQuoteTermsList	LookupList	Y	-	-	-
lookupQuoteSource	LookupList	-	Y	NT	Y
lookupQuoteSourceList	LookupList	Y	-	-	-

lookupQuoteDelivery	LookupList	-	Y	NT	Y
lookupQuoteDeliveryList	LookupList	Y	-	-	-
lookupQuoteCondition	LookupList	-	Y	NT	Y
lookupQuoteConditionList	LookupList	Y	-	-	-
lookupQuoteRemarks	LookupList	-	Y	NT	Y
lookupQuoteRemarksList	LookupList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A **NT** =Not Tested

LookupList

```
code
description
level
enabled
entered
modified
```

XML API: Lookup Lists for Campaigns,Tasks &Action Plans

The following Lookups for Campaigns,Tasks & Action Plans objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
lookupSurveyTypes	LookupList	-	Y	NT	Y
lookupSurveyTypesList	LookupList	Y	-	-	-
lookupTaskPriority	LookupList	-	Y	NT	Y
lookupTaskPriorityList	LookupList	Y	-	-	-
lookupTaskLoe	LookupList	-	Y	NT	Y
lookupTaskLoeList	LookupList	Y	-	-	-
lookupTicketTaskCategory	LookupList	-	Y	NT	Y

lookupTicketTaskCategoryList	LookupList	Y	-	-	-
lookupDurationType	LookupList	-	Y	NT	Y
lookupDurationTypeList	LookupList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A **NT** =Not Tested

LookupList

```
code
description
level
enabled
entered
modified
```

XML API: Custom Lookup Lists

XML API: Custom Lookup Lists for Users,Accounts,Contacts

The following Custom lookups for Accounts ,Contacts objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
lookupIndustryList	CustomLookupList	Y	-	-	-
lookupSiteIdList	CustomLookupList	Y	-	-	-
lookupStageList	CustomLookupList	Y	-	-	-
lookupContactTypesList	CustomLookupList	Y	-	-	-
lookupSubSegmentList	CustomLookupList	Y	-	-	-
contactLeadSkippedMapList	CustomLookupList	Y	-	-	-
contactLeadReadMapList	CustomLookupList	Y	-	-	-

cfsNoteLinkList	CustomLookupList	Y	-	-	-
contactTypeLevelsList	CustomLookupList	Y	-	-	-
lookupListsLookupList	CustomLookupList	Y	-	-	-
categoryEditorLookupList	CustomLookupList	Y	-	-	-
userGroupMapList	CustomLookupList	Y	-	-	-
lookupCallPriorityList	CustomLookupList	Y	-	-	-
lookupCallReminderList	CustomLookupList	Y	-	-	-
lookupCallResultList	CustomLookupList	Y	-	-	-

lookupOpportunityTypesList	CustomLookupList	Y	-	-	-
opportunityComponentLevelsList	CustomLookupList	Y	-	-	-
lookupAccessTypesList	CustomLookupList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A

XML API: Custom Lookup Lists for Projects

The following Custom lookups for Projects objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
lookupProjectActivityList	CustomLookupList	Y	-	-	-
lookupProjectPriorityList	CustomLookupList	Y	-	-	-
lookupProjectStatusList	CustomLookupList	Y	-	-	-
lookupProjectLoeList	CustomLookupList	Y	-	-	-
lookupProjectRoleList	CustomLookupList	Y	-	-	-
lookupProjectCategoryList	CustomLookupList	Y	-	-	-
lookupNewsTemplateList	CustomLookupList	Y	-	-	-
projectPermissionsList	CustomLookupList	Y	-	-	-

projectAccountsList	CustomLookupList	Y	-	-	-
projectTicketCountList	CustomLookupList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A

XML API: Custom Lookup Lists for Products

The following Custom lookups for Products objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
packageProductsMapList	CustomLookupList	Y	-	-	-
productCatalogCategoryMapList	CustomLookupList	Y	-	-	-
productOptionMapList	CustomLookupList	Y	-	-	-
productOptionBooleanList	CustomLookupList	Y	-	-	-
productOptionFloatList	CustomLookupList	Y	-	-	-
productOptionTimestampList	CustomLookupList	Y	-	-	-
productOptionIntegerList	CustomLookupList	Y	-	-	-

productOptionTextList	CustomLookupList	Y	-	-	-
productKeywordMapList	CustomLookupList	Y	-	-	-
productCategoryMapList	CustomLookupList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A

XML API: Custom Lookup Lists for tickets & quotes

The following Custom lookups for tickets & quotes objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
ticketPriorityList	CustomLookupList	Y	-	-	-
ticketLinkProjectList	CustomLookupList	Y	-	-	-
quoteGroupList	CustomLookupList	Y	-	-	-
quoteProductOptionBooleanList	CustomLookupList	Y	-	-	-
quoteProductOptionFloatList	CustomLookupList	Y	-	-	-
quoteProductOptionTimestampList	CustomLookupList	Y	-	-	-
quoteProductOptionIntegerList	CustomLookupList	Y	-	-	-

quoteProductOptionTextList	CustomLookupList	Y	-	-	-
ticketSeverityList	CustomLookupList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A

XML API: Custom Lookup Lists for Communications & Custom Fields

The following Custom lookups for tickets & quotes objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
customFieldLookupList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-
excludedRecipientList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-
campaignListGroupsList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-
activeCampaignGroupsList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-
campaignSurveyLinkList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-

activeSurveyAnswerAvgList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-
fieldTypesList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-
searchFieldElementList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-
messageTemplateList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-
savedCriteriaElementList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-
taskLinkContactList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-

taskLinkTicketList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-
taskLinkProjectList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-
taskCategoryProjectList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-
taskCategoryLinkNewsList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-
moduleFieldCategoryLinkList	org.aspcfs.utils.web.CustomLookupList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A

XML API: Custom Lookup Lists for Document Stores & Action Plans

The following Custom lookups for Document Stores & Action Plans objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
documentStoreRoleList	CustomLookupList	Y	-	-	-
documentStorePermissionList	CustomLookupList	Y	-	-	-
documentStorePermissionsList	CustomLookupList	Y	-	-	-
lookupStepActionsList	CustomLookupList	Y	-	-	-
actionPlanConstantsList	CustomLookupList	Y	-	-	-
stepActionMapList	CustomLookupList	Y	-	-	-
actionStepAccountTypesList	CustomLookupList	Y	-	-	-

documentStorePermissionCategoryList	CustomLookupList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A

XML API: Accounts

The following objects can be accessed by the API:

Id	Mapped Class	S	I	U	D
account	Organization	-	Y	Y	Y
accountList	Organization List	Y	-	-	-
organization Address	Organization Address	-	Y	Y	Y
organization AddressList	Organization AddressList	Y	-	-	-
organization EmailAddres s	Organization EmailAddres s	-	Y	Y	Y
organization EmailAddres sList	Organization EmailAddres sList	Y	-	-	-
organization PhoneNumb er	Organization PhoneNumb er	-	Y	Y	Y
organization PhoneNumb erList	Organization PhoneNumb erList	Y	-	-	-

S = Select **I** = Insert **U** = Update **D** = Delete **Y** = Supported **-** = N/A

Organization

```
orgId (guid)
name
accountNumber
url
revenue
employees
notes
ticker
entered
enteredBy (references:[[XML API: Users, Roles & Permissions|user]])
modified
modifiedBy (references:[[XML API: Users, Roles & Permissions|user]])
enabled
industry (references: lookupIndustry)
owner (references:[[XML API: Users, Roles & Permissions|user]])
contractEndDate
alertDate
alertText
nameSalutation
nameLast
nameFirst
nameMiddle
nameSuffix
importId
statusId
alertDateTimeZone
contractEndDateTimeZone
trashedDate
source (references: lookupContactSource)
rating (references: lookupContactRating)
potential
segmentId (references: lookupSegments)
subSegmentId (references: lookupSubSegment)
accountSize (references: lookupAccountSize)
siteId (references: lookupSiteId)
businessNameTwo
yearStarted
stageId (references: lookupAccountStage)
```

OrganizationAddress

```
id (guid)
orgId (references: account)
type (references: lookupOrgAddressTypes)
streetAddressLine1
streetAddressLine2
streetAddressLine3
streetAddressLine4
city
state
country
zip
entered
enteredBy (references: [[XML API: Users, Roles & Permissions|user]])
modified
modifiedBy (references: [[XML API: Users, Roles & Permissions|user]])
primaryAddress
county
latitude
longitude
```

OrganizationEmailAddress

```
id (guid)
orgId (references: account)
type (references: lookupOrgEmailTypes)
email
enteredBy (references: [[XML API: Users, Roles & Permissions|user]])
modifiedBy (references: [[XML API: Users, Roles & Permissions|user]])
entered
modified
primaryEmail
```

OrganizationPhoneNumber

```
id (guid)
orgId (references: account)
number
extension
type (references: lookupOrgPhoneTypes)
enteredBy (references: [[XML API: Users, Roles & Permissions|user]])
modifiedBy (references: [[XML API: Users, Roles & Permissions|user]])
entered
modified
primaryNumber
```

Tools Package

ConcourseSuite CRM Tools is a small java library (.jar) which encapsulates the HTTP and XML code so that it's easier to write applications that talk with Concursive's products. For example, you can capture leads or tickets from your existing web site and send them straight into ConcourseSuite CRM by using the library's DataRecord and "save" action method.

You can also read data from the CRM by using the "load" method. The following actions are supported:

- DataRecord.INSERT
- DataRecord.SELECT
- DataRecord.UPDATE
- DataRecord.DELETE
- DataRecord.GET_DATETIME

Requirements

The crm_tools.jar can be used with any Java 1.5 or newer application. You will need to have the Apache Commons Codec-API in your classpath.

A "client" will need to be configured under ConcourseSuite's Admin module to provide remote access to CRM data.

The client records are located in the database and can be modified manually:

- In the [sync_client] table, an arbitrary client should be inserted with a plain-text password in the [code] field. This will be used in the client authentication code.

Typical Usage

```
import com.concursive.crm.api.client.CRMConnection;
import com.concursive.crm.api.client.DataRecord;

// Client ID must already exist in target CRM system and is created
// under Admin -> Configure System -> HTTP-XML API Client Manager
int clientId = 1;

// Establish connectivity information
CRMConnection crm = new CRMConnection();
crm.setUrl("http://www.example.com/crm");
crm.setId("www.example.com");
crm.setCode("password");
crm.setClientId(clientId);

// Start a new transaction
crm.setAutoCommit(false);

DataRecord contact = new DataRecord();
contact.setName("contact");
contact.setAction(DataRecord.INSERT);
contact.setShareKey(true);
contact.addField("nameFirst", bean.getNameFirst());
contact.addField("nameLast", bean.getNameLast());
contact.addField("company", bean.getCompanyName());
contact.addField("title", bean.getTitle());
contact.addField("source", bean.getSourceId());
contact.addField("isLead", "true");
contact.addField("accessType", 2);
contact.addField("leadStatus", 1);
contact.addField("enteredBy", 0);
contact.addField("modifiedBy", 0);
crm.save(contact);

// Transform the email
DataRecord email = new DataRecord();
email.setName("contactEmailAddress");
email.setAction(DataRecord.INSERT);
email.addField("email", bean.getEmail());
email.addField("contactId", "$C{contact.id}");
email.addField("type", 1);
email.addField("enteredBy", 0);
email.addField("modifiedBy", 0);
crm.save(email);

// Transform the phone
DataRecord phone = new DataRecord();
phone.setName("contactPhoneNumber");
phone.setAction(DataRecord.INSERT);
phone.addField("number", bean.getPhone());
phone.addField("contactId", "$C{contact.id}");
phone.addField("type", 1);
phone.addField("enteredBy", 0);
phone.addField("modifiedBy", 0);
crm.save(phone);

boolean result = crm.commit();

System.out.println(crm.getLastResponse());
```

XML API for PHP

Using the XML API, data in ConcourseSuite CRM can be retrieved using various clients. While a toolset has not been made specifically for PHP, the following has been used to interact with ConcourseSuite CRM and PHP.

In the example, the XML creation is left up to the application to implement, although an XML library is strongly encouraged so that the XML is encoded properly.

```
<?
// Site to post to
$server = "www.example.com"
$path = "/crm/ProcessPacket.do"
// Begin the transfer
$socket = fsockopen($server, 80);
if (!$socket) {
// error: could not connect to server
// return
} else {
// Post the XML document
$request = "POST " . $path . " HTTP/1.0\r\n" .
"Host: " . $server . "\r\n" .
"Content-Type: text/xml\r\n" .
"Content-Length: " . strlen($xmlDocument) . "\r\n\r\n" .
$xmlDocument;
if (!fputs($socket, $request, strlen($request))) {
// error: could not write to server
// return
}
$response = '';
while ($data = fread($socket, 32768)) {
$response .= $data;
}
fclose($socket);
if ($response = '') {
// error: no response from server
// return
} else {
// review the response for status
}
}
?>
```

Some things to keep in mind...

- The XML needs to be well-formed
- Errors in the response provide some answers... see the FAQ

XML API Examples

The following examples provide an overview of the XML API use. Each example provides both the code snippet that can used with the Tools Library or the actual xml sent as part of the request packet.

QUERY

1. Fetch a list of ALL Accounts with details requested
2. Fetch all Accounts OWNED by a particular user
3. Fetch a list of lookup items

ADD

1. Add Contact & related information
2. Add an Opportunity and an associated Opportunity Component

SYNC

1. Sync a list of Accounts within a particular time period

Fetch a list of ALL Accounts with details requested

Using Centric Tools

```
// Establish connectivity information
CRMConnection crm = new CRMConnection();
crm.setUrl("http://www.example.com/crm");
crm.setId("www.example.com");
crm.setCode("password");
crm.setClientId(clientId);

//Add Meta Info with fields required
ArrayList meta = new ArrayList();
meta.add("orgId");
meta.add("name");
meta.add("url");
meta.add("notes");
meta.add("industryName");
meta.add("alertDate");
meta.add("alertText");
meta.add("revenue");
meta.add("ticker");
meta.add("accountNumber");
meta.add("potential");
meta.add("nameFirst");
meta.add("nameMiddle");
meta.add("nameLast");
crm.setTransactionMeta(meta);

DataRecord accountsTable = new DataRecord();
accountsTable.setName("accountList");
accountsTable.setAction(DataRecord.SELECT);

crm.load(accountsTable);
```

XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<app>
  <authentication>
    <id>www.example.com</id>
    <systemId>4</systemId>
    < code>+++ CLIENT'S PASSWORD +++< /code>
    <clientId>+++ CLIENT ID ALREADY ESTABLISHED +++</clientId>
  </authentication>
  <transaction id="1">
    <meta>
      <property>orgId</property>
      <property>name</property>
      <property>url</property>
      <property>notes</property>
      <property>industryName</property>
      <property>alertDate</property>
      <property>alertText</property>
      <property>revenue</property>
      <property>ticker</property>
      <property>accountNumber</property>
      <property>potential</property>
      <property>nameFirst</property>
      <property>nameMiddle</property>
      <property>nameLast</property>
    </meta>
    <accountList action="select"/>
  </transaction>
</app>
```

Fetch all Accounts OWNED by a particular user

Using Centric Tools

```
// Establish connectivity information
CRMConnection crm = new CRMConnection();
crm.setUrl("http://www.example.com/crm");
crm.setId("www.example.com");
crm.setCode("password");
crm.setClientId(clientId);

//Add Meta Info with fields required
ArrayList meta = new ArrayList();
meta.add("orgId");
meta.add("name");
meta.add("url");
meta.add("notes");
crm.setTransactionMeta(meta);

DataRecord accountsTable = new DataRecord();
accountsTable.setName("accountList");
accountsTable.setAction(DataRecord.SELECT);
accountsTable.addField("ownerId", +++ USER ID +++);
crm.load(accountsTable);
```

XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<app>
  <authentication>
    <id>www.yourorg.com</id>
    <systemId>4</systemId>
    < code>+++ CLIENT'S PASSWORD +++< /code>
    <clientId>+++ CLIENT ID ALREADY ESTABLISHED +++</clientId>
  </authentication>
  <transaction id="1">
    <meta>
      <property>orgId</property>
      <property>name</property>
      <property>url</property>
      <property>notes</property>
    </meta>
    <accountList action="select">
      <ownerId>+++ USER ID +++</ownerId>
    </accountList>
  </transaction>
</app>
```

Fetch a list of lookup items

Using Centric Tools

```
// Establish connectivity information
CRMConnection crm = new CRMConnection();
crm.setUrl("http://www.example.com/crm");
crm.setId("www.example.com");
crm.setCode("password");
crm.setClientId(clientId);

//Add Meta Info with fields required
ArrayList meta = new ArrayList();
meta.add("code");
meta.add("description");
crm.setTransactionMeta(meta);

DataRecord lookupIndustryTypes = new DataRecord();
lookupIndustryTypes.setName("lookupIndustryList");
lookupIndustryTypes.setAction(DataRecord.SELECT);
lookupIndustryTypes.addField("tableName", "lookup_industry");
lookupIndustryTypes.addField("uniqueField", "code");
lookupIndustryTypes.addField("property", "code");
lookupIndustryTypes.addField("property", "description");
crm.load(lookupIndustryTypes);
```

XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<app>
  <authentication>
    <id>www.example.com</id>
    <systemId>4</systemId>
    <code>+++ CLIENT'S PASSWORD +++</code>
    <clientId>+++ CLIENT ID ALREADY ESTABLISHED +++</clientId>
  </authentication>
  <transaction id="1">
    <meta>
      <property>code</property>
      <property>description</property>
    </meta>
    <lookupIndustryList action="select">
      <tableName>lookup_industry</tableName>
      <uniqueField>code</uniqueField>
      <property>code</property>
      <property>description</property>
    </lookupIndustryList>
  </transaction>
</app>
```

Add Contact & related information

Using Centric Tools

```
// Establish connectivity information
CRMConnection crm = new CRMConnection();
crm.setUrl("http://www.example.com/crm");
crm.setId("www.example.com");
crm.setCode("password");
crm.setClientId(clientId);

DataRecord contact = new DataRecord();
contact.setName("contact");
contact.setAction(DataRecord.INSERT);
contact.setShareKey(true);
contact.addField("nameFirst", bean.getNameFirst());
contact.addField("nameLast", bean.getNameLast());
contact.addField("company", bean.getCompanyName());
contact.addField("title", bean.getTitle());
contact.addField("source", bean.getSourceId());
contact.addField("isLead", "true");
contact.addField("accessType", 2);
contact.addField("leadStatus", 1);
contact.addField("enteredBy", 0);
contact.addField("modifiedBy", 0);
crm.save(contact);

//email
DataRecord email = new DataRecord();
email.setName("contactEmailAddress");
email.setAction(DataRecord.INSERT);
email.addField("email", bean.getEmail());
email.addField("contactId", "$C{contact.id}");
email.addField("type", 1);
email.addField("enteredBy", 0);
email.addField("modifiedBy", 0);
crm.save(email);

//phone
DataRecord phone = new DataRecord();
phone.setName("contactPhoneNumber");
phone.setAction(DataRecord.INSERT);
phone.addField("number", bean.getPhone());
phone.addField("contactId", "$C{contact.id}");
phone.addField("type", 1);
phone.addField("enteredBy", 0);
phone.addField("modifiedBy", 0);
crm.save(phone);

DataRecord address = new DataRecord();
address.setName("contactAddress");
address.setAction(DataRecord.INSERT);
address.addField("streetAddressLine1", bean.getStreetAddressLine1());
address.addField("streetAddressLine2", bean.getStreetAddressLine2());
address.addField("streetAddressLine3", bean.getStreetAddressLine3());
address.addField("streetAddressLine4", bean.getStreetAddressLine4());
address.addField("contactId", "$C{contact.id}");
address.addField("type", 1);
address.addField("city", bean.getCity());
address.addField("state", bean.getState());
address.addField("zip", bean.getZip());
address.addField("country", bean.getCountry());
address.addField("enteredBy", 0);
address.addField("modifiedBy", 0);
address.addField("primaryAddress", true);
crm.save(address);

boolean result = crm.commit();
```

```
boolean result = crm.commit();
```

XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<app>
  <authentication>
    <id>www.example.com</id>
    <systemId>4</systemId>
    < code>+++ CLIENT'S PASSWORD +++< /code>
    <clientId>+++ CLIENT ID ALREADY ESTABLISHED +++</clientId>
  </authentication>
  <transaction id="1">
    <contact action="insert" shareKey="true">
      <nameFirst>John</nameFirst>
      <nameLast>Doe</nameLast>
      <company>XXX</company>
      <title>XXX</title>
      <accessType>2</accessType>
      <enteredBy>0</enteredBy>
      <modifiedBy>0</modifiedBy>
    </contact>
    <contactEmailAddress action="insert">
      <email>john.doe@xxx.com</email>
      <contactId>$C{contact.id}</contactId>
      <type>1</type>
      <enteredBy>0</enteredBy>
      <modifiedBy>0</modifiedBy>
    </contactEmailAddress>
    <contactPhoneNumber action="insert">
      <number>8888888888</email>
      <contactId>$C{contact.id}</contactId>
      <type>1</type>
      <enteredBy>0</enteredBy>
      <modifiedBy>0</modifiedBy>
    </contactPhoneNumber>
    <contactAddress action="insert">
      <contactId>$C{contact.id}</contactId>
      <type>1</type>
      <streetAddressLine1>200 Yoakum Pkwy</streetAddressLine1>
      <streetAddressLine2>Apt 444</streetAddressLine2>
      <streetAddressLine3></streetAddressLine3>
      <streetAddressLine4></streetAddressLine4>
      <city>Alexandria</city>
      <state>VA</state>
      <zip>22345</zip>
      <country>USA</country>
      <enteredBy>0</enteredBy>
      <modifiedBy>0</modifiedBy>
      <primaryAddress>true</primaryAddress>
    </contactAddress>
  </transaction>
</app>
```

Add an Opportunity and an associated Opportunity Component

Using Centric Tools

```
// Establish connectivity information
CRMConnection crm = new CRMConnection();
crm.setUrl("http://www.example.com/crm");
crm.setId("www.example.com");
crm.setCode("password");
crm.setClientId(clientId);

//Add Meta Info with fields required
ArrayList meta = new ArrayList();
meta.add("description");
crm.setTransactionMeta(meta);

DataRecord opportunity = new DataRecord();
opportunity.setName("opportunity");
opportunity.setAction(DataRecord.INSERT);
opportunity.setShareKey(true);
opportunity.addField("description", "Opportunity Header");
opportunity.addField("manager", 1); //refers to user with id 1 in CCRM
opportunity.addField("accountLink", 1); //refers to account with id 1 in CCRM
opportunity.addField("accessType", 9); //indicates it is a Public opportunity
record
opportunity.addField("enteredBy", 1);
opportunity.addField("modifiedBy", 1);
crm.save(opportunity);

DataRecord oppComponent = new DataRecord();
oppComponent.setName("opportunity");
oppComponent.setAction(DataRecord.INSERT);
oppComponent.addField("description", "Opportunity Component");
oppComponent.addField("owner", 1); //refers to user with id 1 in CCRM
oppComponent.addField("guess", 800000);
oppComponent.addField("type", "N"); //indicates it is a new opportunity component;
corresponds to 'Source' on the add form
oppComponent.addField("stage", 1); //corresponds to 'Prospecting' lookup entry in
lookup_stage
oppComponent.addField("closeProb", 50);
oppComponent.addField("closeDate", "09/14/2007");
oppComponent.addField("closeDateTimeZone", "GMT-5 Easter US");
oppComponent.addField("terms", "15"); //indicates the time period
oppComponent.addField("units", "W"); //indicates the time period units; W for
Weeks; M for Months
oppComponent.addField("enteredBy", 1); //refers to user with id 1 in CCRM
oppComponent.addField("modifiedBy", 1); //refers to user with id 1 in CCRM
crm.save(oppComponent);

boolean result = crm.commit();
```

XML

```
<app>
  <authentication>
    <id>127.0.0.1</id>
    <systemId>4</systemId>
    <clientId>1</clientId>
    < code>792fef441691aa41135a15c1478a5ee4< /code>
  </authentication>
  <transaction>
    <meta>
      <property>description</property>
    </meta>
    <opportunity action="insert" shareKey="true">
      <description>Opportunity Header</description>
      <manager>1</manager>
      <accountLink>1</accountLink>
      <accessType>9</accessType>
      <enteredBy>0</enteredBy>
      <modifiedBy>0</modifiedBy>
    </opportunity>
    <opportunityComponent action="insert">
      <headerId>$C{opportunity.id}</headerId>
      <owner>1</owner>
      <description>Component Description</description>
      <guess>80</guess>
      <type>N</type>
      <stage>1</stage>
      <closeProb>50</closeProb>
      <closeDate>09/14/2007</closeDate>
      <closeDateTimeZone>"GMT-5 Eastern US"</closeDateTimeZone>
      <terms>15</terms>
      <units>W</units>
      <enteredBy>0</enteredBy>
      <modifiedBy>0</modifiedBy>
    </opportunityComponent>
  </transaction>
</app>
```

Sync a list of Accounts within a particular time period

Using Centric Tools

```
// Establish connectivity information
CRMConnection crm = new CRMConnection();
crm.setUrl("http://www.example.com/crm");
crm.setId("www.example.com");
crm.setCode("password");
crm.setClientId(clientId);
//Specify anchors
crm.setLastAnchor("2007-09-02 12:00:00.000");
crm.setNextAnchor("2007-09-22 12:00:00.000");

//Add Meta Info with fields required
ArrayList meta = new ArrayList();
meta.add("name");
meta.add("accountNumber");
meta.add("contractEndDate");
crm.setTransactionMeta(meta);

DataRecord accounts = new DataRecord();
accounts.setName("accountList");
accounts.setAction(DataRecord.SYNC);
crm.load(accounts);
```

XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<app>
  <authentication>
    <id>127.0.0.1</id>
    <systemId>4</systemId>
    <clientId>1</clientId>
    < code>792fef441691aa41135a15c1478a5ee4< /code>
    <lastAnchor>2007-09-02 12:00:00.000</lastAnchor>
    <nextAnchor>2007-09-22 12:00:00.000</nextAnchor>
  </authentication>
  <transaction>
    <meta>
      <property>name</property>
      <property>accountNumber</property>
      <property>contractEndDate</property>
    </meta>
    <accountList action="sync"/>
  </transaction>
</app>
```

FAQ

Server Responses

1. When I send an XML Packet to the server, I get "Not Authorized" response?

2. I am sending an UPDATE request and I receive a "conflict" in the response?

SELECT API Call

1. I need to restrict the records sent from the server by setting some filters. How do I do that?

2. I am setting some filters in the SELECT api call, but the server is sending all the records back. Why?

I am sending an UPDATE request and I receive a "conflict" in the response?

When you call an "UPDATE" method, the corresponding update method in the object specified is called to execute an UPDATE SQL statement. The sql uses the 'modified' sent in the xml and compares it with the 'modified' that currently exists in the database for this object. The reason for this comparison is to avoid 2 users simultaneously updating the record and also to avoid someone overwriting the record with stale data.

Let me try to provide an example for the stale data. Say you and I use the XML api to query the same "Task" object today. Now on both of our individual remote systems the task object exists as it was today. Say tomorrow I send an UPDATE request XML packet for this task. If it was updated successfully then the version of the task on the server and on my system match. Your system would have a stale version of the task. Now if you try to update the task without looking at the latest values available for this task at the server, then the server should reject your task UPDATE packet.

It would be the responsibility of the client using the API to keep track of the 'modified' timestamp against the object that was fetched from the server. Whenever the client tries to update the object, it should send the 'modified' value that it has. If the object at the server has already been updated by someone else then the 'modified' will no longer match and server will report a conflict, in which case the client should first query the object for the latest values (including modified timestamp of the object at the server) and send the update back which would then work.

So in your packet please include the <modified>[timestamp that you had fetched using a select in the past for this object]</modified>

I need to restrict the records sent from the server by setting some filters. How do I do that?

Filters that can be set for a SELECT XML API call should be properties that are defined in the list class (Contact email addresses corresponds to ContactEmailAddressList which extends EmailAddressList). These properties are used in the java source files to restrict the records. Also the name specified in the API call should match the member present in the List class. There is no documentation which describes the filters available in each List class that can be applied. Hence the developer will need to look at the CRM source code to determine the fields. The documentation will be updated in the future to specify what filters are available for a particular class.

Here is an XML API example where a list of contact email addresses are requested from the server, but returned records need to belong to a particular contact.

```
//Add Meta Info
ArrayList meta = new ArrayList();
meta.add("email");
meta.add("typeName");
meta.add("primaryEmail");
crm.setTransactionMeta(meta);

DataRecord addresses = new DataRecord();
addresses.setName("contactEmailAddressList");
addresses.setAction(DataRecord.SELECT);
addresses.addField("contactId", *** SPECIFY CONTACT ID ***); //filter addresses for
a particular contact
crm.load(addresses);

boolean result = crm.commit();
System.out.println("RESPONSE: " + crm.getLastResponse());

Object[] objects =
crm.getRecords("com.concursive.crm.web.modules.contacts.dao.ContactEmailAddress").t
oArray();
ContactEmailAddress[] addresses = new ContactEmailAddress[objects.length];
for (int i = 0; i < objects.length; i++) {
  addresses[i] = (ContactEmailAddress) objects[i];
}
```

Here is a snapshot of what the com.concursive.crm.web.modules.contacts.dao.EmailAddressList Class looks like

```
public class EmailAddressList extends Vector {
   ......
  protected int orgId = -1;
  protected int type = -1;
  protected int contactId = -1;
  protected String username = null;
  protected String emailAddress = null;
  //instance info
  protected int instanceId = -1;

   ......

   public void setContactId(int tmp) {
      this.contactId = tmp;
   }

   public void setContactId(String tmp) {
      this.contactId = Integer.parseInt(tmp);
   }
```

When the API receives the xml packet, it sets the 'contactId' filter and populates the ContactEmailAddressList object, which results in a list of ContactEmailAddress objects whose contactId matches the one specified by the user. The other properties can also be set for filtering the addresses.

I am setting some filters in the SELECT api call, but the server is sending all the records back. Why?

To Remember...

1. Filters that can be set for a SELECT XML API call should be properties that are defined in the list class (Contact email addresses corresponds to ContactEmailAddressList which extends EmailAddressList). These properties are used in the java source files to restrict the records. Also the name specified in the API call should match the member present in the List class. There is no documentation which describes the filters available in each List class that can be applied. Hence the developer will need to look at the CRM source code to determine the fields. The documentation will be updated in the future to specify what filters are available for a particular class.
2. The API reads the filter specified in the call and tries to set it on the list object (eg: ContactEmailAddressList). If the filter was incorrectly spelled or is not defined in the List class and is not used in the query available in the List class, then the ContactEmailAddressList will get executed with no filters and will hence return all the records. This is the way the API works today.

Importing Data

A list of people and companies can be imported in several ways. During the installation of a system, the most common are importing .csv files and programmatically using the Java reader/writer applications.

Importing .CSV Files

The Leads, Contacts, and Accounts modules have a web-based import capability. If the user has permission, then a list of people and/or organizations can be imported using the web-based wizard.

The import tool will associate multiple contacts with the same organization name to the same account.

Including column names in the first row makes it much simpler to manage field mappings, as shown below.

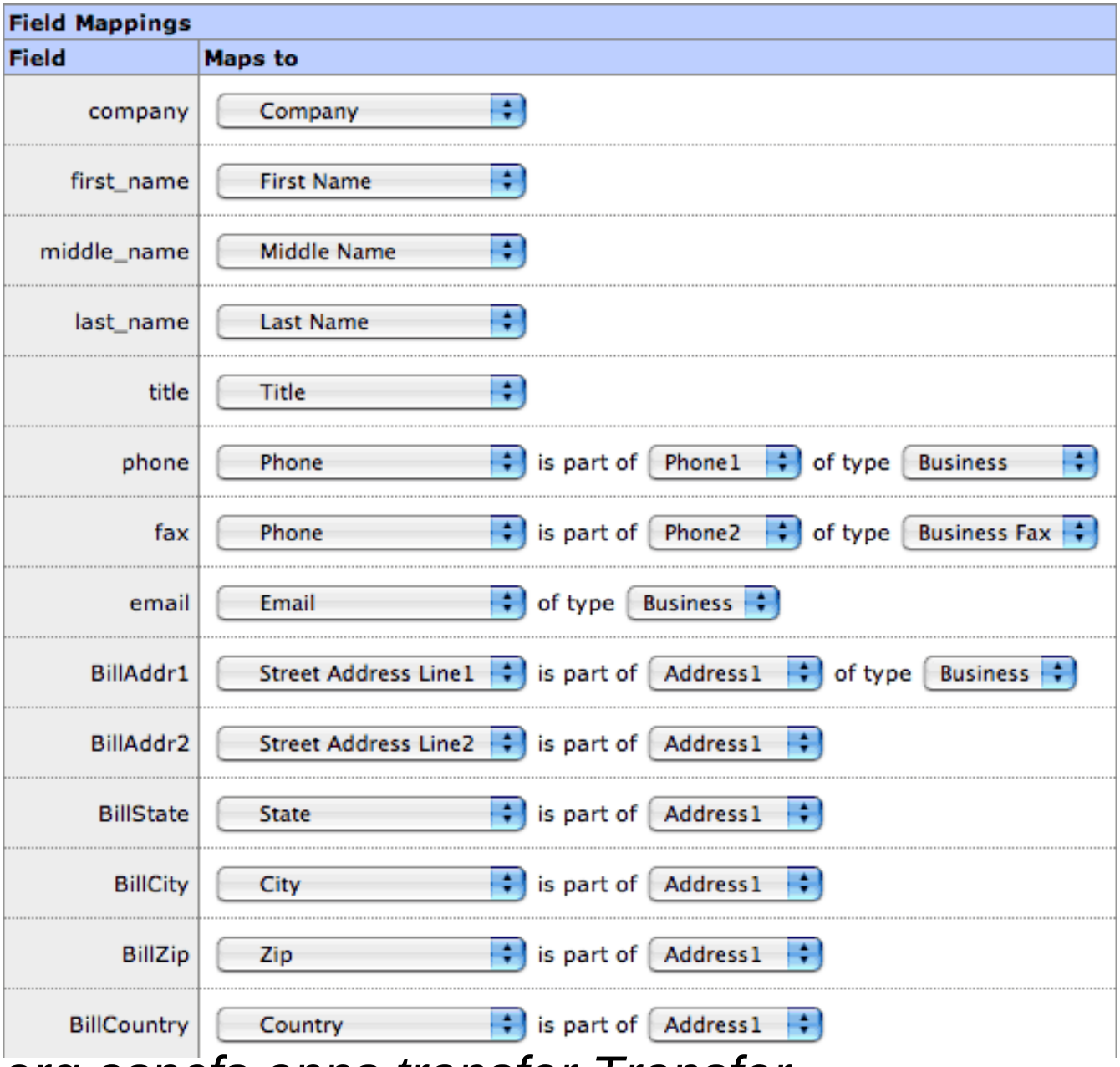
Using Reader/Writer

The Reader/Writer applications can read data from various input formats and write data to various output formats. These applications can interface with files, databases, web services, etc.

To import related data, for example Accounts, Contacts, Opportunities, and Notes, a Reader can be developed and executed.

The tools include a Writer that knows how to write data into the CRM, and comes with some basic Readers.

Readers are Java programs that send data to a Writer and associates data during the reading/writing process. The reader and writer are configured and executed by the Transfer application.



org.aspcfs.apps.transfer.Transfer

The Transfer application begins the process of migrating data from a DataReader to a DataWriter.

Transfer is responsible for loading configuration data, instantiating objects, and executing and monitoring the data import process.

The following is included in the Centric CRM build.xml file, which can be altered and used for importing Account Contacts and related data...

```
  <!-- Account Contact Importer based on CSV spec -->
  <target name="import.accountcontacts" depends="init">
    <fail unless="arg1">Missing arg1: CSV file to import not specified</fail>
    <fail unless="arg2">Missing arg2: Virtual Host to import to not
specified</fail>
    <fail unless="arg3">Missing arg3: Import code not specified</fail>
    <mkdir dir="${build.pref.dir}"/>
    <copy file="${pref.dir}/cfs/transfer/import-accountcontacts.xml"
      todir="${build.pref.dir}" overwrite="true"/>
    <replace file="${build.pref.dir}/import-accountcontacts.xml"
      token="@PROPERTY.FILE@" value="${pref.dir}/cfs/transfer/import-
mappings.xml"/>
    <replace file="${build.pref.dir}/import-accountcontacts.xml"
      token="@CSV.FILE@" value="${arg1}"/>
    <replace file="${build.pref.dir}/import-accountcontacts.xml"
      token="@URL@" value="${arg2}"/>
    <replace file="${build.pref.dir}/import-accountcontacts.xml"
      token="@ID@" value="${arg3}"/>
    <replace file="${build.pref.dir}/import-accountcontacts.xml"
      token="@CODE@" value="${arg4}"/>
    <replace file="${build.pref.dir}/import-accountcontacts.xml"
      token="@SYSTEM.ID@" value="4"/>
    <java classname="org.aspcfs.apps.transfer.Transfer" fork="yes"
failonerror="yes">
      <classpath>
        <path refid="cfs2.classpath"/>
        <fileset dir="${CENTRIC_HOME}/WEB-INF/lib">
          <include name="**/*.jar"/>
        </fileset>
      </classpath>
      <arg value="${build.pref.dir}${fs}import-accountcontacts.xml"/>
    </java>
  </target>
```

Development Process

ConcourseSuite development includes making enhancements and changes to the core web application framework, modules, components, supporting applications, database schemas, and installation and upgrade scripts.

In order to benefit all community members, a shared, collaborative process provides up-to-date project status and documentation. Members can see what features are being considered and developed, as well as contribute insight and expertise.

The following Project Management web site tools are extensively used throughout the ConcourseSuite project:

- Project announcements and news to stay up-to-date with current project events
- Resource allocation and contact management to see who is working on what
- Reference information and specifications
- Discussion groups for sharing ideas and gathering feedback
- Issue tracking for reporting and working through defects and enhancements
- Document repository for storing related project material
- Source code repository (Subversion) for centralizing code

Development begins with a well-understood project description, then typically steps through user needs, planning, specifications, design elements, code development, review, QA, testing, documentation and installation.

Use the following steps to ensure a successful contribution to the code.

Well-Understood Project Description

By preparing a well-understood project description, you will be certain to stay on track and allow others to understand what you are setting out to accomplish.

The description should include an introduction of what is being proposed, the goals that are to be achieved with this project, the scope of work to be done, and a high-level outline of deliverables.

Legal Issues

Any legal issues related to the project should be documented. These can be in reference to code ownership or regulatory compliance.

Remember, code contributions to ConcourseSuite cannot use GPL code or GPL libraries due to licensing restrictions. Typical acceptable open-source licenses include Apache Software License, LGPL and BSD.

User Needs

Next, outline the business, system, and server environment in which the software will be used. For example, the implementation of a sophisticated graphing system might be demanding on processor power. However, if this process is only expected to be executed once per night, then the specifications might not require a queue.

Make sure to outline the stakeholders and the actors. These can be named individuals or roles that they play. List their needs and what they have in common. It's important to understand who will be using the system and why.

Specifications will be driven by these expectations.

Software Specifications

Before coding can truly begin, specifications need to be generated and agreed upon.

1. **Use Cases** -- Specify how the stakeholders and actors will interact with the system. You should already know who these users are and why they need to use the system, but now you will show exactly what they can do in the system. These will include use-cases by functional area and by priority. Associate business objects with the roles as well.
2. **Functional Requirements** -- List the feature set and implementation priorities.
3. **Non-Functional Requirements** -- List the items that will have an impact and explain how to resolve them.
    - Reliability/uptime
    - Security
    - Performance and scalability
    - Maintainability and upgradeability
4. **Environmental Requirements**

- Hardware
- Software
- Programming Interfaces
- Data import/export

Project Plan

Once the scope of the project is understood, the work to be done should be broken down into sections with time estimates so that a schedule can be devised. Risks that might impact the project should also be specified.

A project plan will serve as an outline for assessing milestones.

Design

1. System Architecture: components, site-map
2. User Interface: content model, wireframes, prototypes
3. Persistence/Database Schema
4. Security Model

QA

1. Identify quality goals and rate them by priority
    - Functionality
    - Usability
    - Security
    - Reliability
    - Efficiency
    - Scalability
    - Operability
    - Maintainability
2. Implement a QA Strategy
    - Community review
    - Unit testing
    - System testing

Testing

1. Business objects and operation
2. Features
3. Use-cases

Documentation

1. Installation/Upgrade
2. Release notes
3. User Guide

Upgrade and Installation

Developer References

The following links will provide you with relevant information on various subjects that relate to Centric CRM and Java development.

- The Java Language Specification
- The Java Tutorial
- Sun Guide to Writing JavaDoc Comments
- Java2 Platform Standard Edition API Specification
- Java2 Platform Enterprise Edition API Specification for Servlets and JSPs
- JSP Documentation

Additional Resources:

- PostgreSQL Documentation and Manuals
- Subversion Book

Please review the Coding Best Practices section for various tips and reminders about coding against the Centric CRM framework.

Coding Best Practices

The following tips will help achieve the best performance, the best security, and the easiest maintenance...

These are from the experts:

- At most, obtain a single connection from the database connection pool; multiple connections can result in a deadlock
- Instead of using servletContext.getRealPath("/"), use servletContext.getResource("/"); This avoids errors when containers return null
- Assume that no data can be written to the deployed webapp directory; instead use the fileLibrary and write servlet actions to stream data to and from the location
- Avoid thread locking in loops -- make the variable that the loop depends upon "volatile"; additionally add a sleep in the waiting loop

Installation

The installation, configuration, maintenance and upgrade of ConcourseSuite is intended to be as simple as possible.

The following information is being copied from the "CRM Installation, Setup and Maintenance" Guide that is available with the the ConcourseSuite download. It's currently incomplete here...

- Steps for a successful Linux Installation
- Steps for a successful Mac OSX Installation
- Steps for a successful Windows Installation
- Setting up a Database Server
- Additional installation steps to consider

Steps for a successful Linux Installation

1. Download and install Oracle Java JRE or JDK 6.0; gcc-java does not work and is bundled with some Linux distributions; The latest versions of ConcourseSuite work with Java 7 and Tomcat 7. ConcourseSuite 6.0 is aligned with Java 6 while ConcourseSuite 5.0 is aligned with Tomcat 5.5 up to 5.5.25.

2. Download and install Tomcat stable "Core" from http://tomcat.apache.org ; Apache Tomcat is bundled with some Linux distributions and fails if it was compiled with gcc-java

3. Depending on the total memory of your system, typically set Tomcat's memory to half of the server by setting the following environment variable (directly in the startup.sh file):

```
export JAVA_OPTS="-Xms512 -Xmx512m -XX:PermSize=64m \
-XX:MaxPermSize=128m"
```

4. Enable software graphics rending, or else ConcourseSuite CRM will fail, by setting the following environment variable (directly in the startup.sh file).

```
# For Tomcat 5.5.27 with older CRM versions only
export CATALINA_OPTS="-Djava.awt.headless=true \
-Dorg.apache.jasper.compiler.Parser.STRICT_QUOTE_ESCAPING=false"
```

5. Download and copy the ConcourseSuite CRM application (crm.war) into Tomcat's "webapps" directory

6. Setup a service to automatically have Tomcat start and stop during startup and shutdown, according to Tomcat's documentation; the user that starts Tomcat must have a home directory as this is where Java stores preferences for Java applications

Permissions for the Tomcat User

The Java API tries to store a preference in the user's home directory. If the user that runs Tomcat does not have a home directory, then after a restart the user will be prompted to configure the application again. Choosing the same path reloads the existing configuration.

If this occurs, then in the Tomcat startup script, you can add a variable called CATALINA_OPTS, or use an environment variable, with a directory that the tomcat user does have access to...

CATALINA_OPTS="-Djava.awt.headless=true \

-Djava.util.prefs.userRoot=/opt/concursive/crm/fileLibrary/userPrefs \

-Djava.util.prefs.systemRoot=/opt/concursive/crm/fileLibrary/systemPrefs"

1. java.awt.headless is used for any server generated graphics

2. the other two specify the directory to write the prefs to (and must already exist with the correct permissions).

Notes about gcc-java

GCJ supports most of the Java 1.4 libraries plus some 1.5 additions, but not all. This causes ConcourseSuite CRM to break. If you are using a Linux distribution with gcj then you can take the following steps after installing Sun Java and Apache Tomcat (do not use the versions included with Linux):

1. Overwrite (or rename) the existing bin/java command by creating a soft link to Sun's bin/java
2. Update /etc/init.d/tomcat with the user that should be running Tomcat
3. Make changes to /etc/ant.conf (if using ConcourseSuite CRM source)

Steps for a successful Mac OSX Installation

1. Download and install Tomcat 6 stable "Core" from http://tomcat.apache.org/download-60.cgi

2. Depending on the total memory of your system, typically set Tomcat's memory to half of the server by setting the following environment variable (in startup.sh):

```
export JAVA_OPTS="-Xms512 -Xmx512m -XX:PermSize=64m -XX:MaxPermSize=128m"
```

3. Download and copy the ConcourseSuite CRM binary (crm.war) file from http://www.concursive.com into Tomcat's "webapps" directory

4. Setup a service to automatically have Tomcat start and stop during startup and shutdown

Steps for a successful Windows Installation

1. Download and install the Sun Java 6.0 from http://www.java.com
2. Download and install Tomcat 6 stable "Core" - Windows Executable from http://tomcat.apache.org/download-60.cgi; depending on the total memory of your system, typically set Tomcat's memory to half of the server using Tomcat's configuration editor
3. Using the Windows Services Control Panel, stop the "Apache Tomcat" service
4. Download and copy the ConcourseSuite CRM binary (crm.war) file from http://www.concursive.com into Tomcat's "webapps" directory -- usually located in C:\Program Files\Apache Software\Tomcat\webapps

Setting up a Database Server

PostgreSQL

- PostgreSQL on Linux
- PostgreSQL on Windows

IBM DB2

- IBM DB2 or DB2 Express-C on Windows
- IBM DB2 or DB2 Express-C on Linux

Microsoft SQL Server

- Microsoft SQL Server 2005 or 2005 Express
- Microsoft SQL Server 2000
- Microsoft SQL Server DE (MSDE)

One$DB / DaffodilDB

- One$DB on any platform

Firebird DB

MySQL

Apache Derby

Additional installation steps to consider

1. Change Tomcat port to 80 in server.xml (either integrate with Apache Web Server or turn off Apache Web server if Tomcat will be using port 80)
2. Obtain and install an SSL certificate to run Centric CRM on port 443

Developing a ConcourseSuite Module

Adding a module to ConcourseSuite involves several items depending on the requirements and integration the module has with the rest of the system.

If a new module is being added, start with skeleton code of the Action Class and JSPs, configure the ConcourseSuite framework, then iteratively add functionality.

Adding or updating a module includes the following steps:

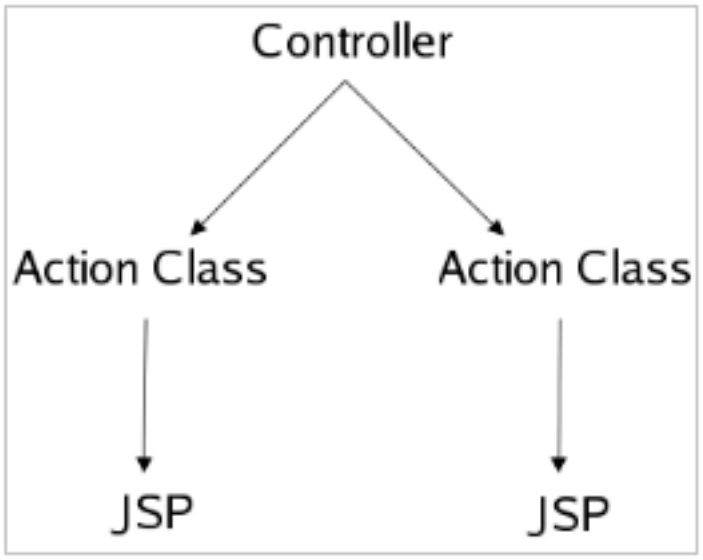
1. Determine the framework model that best fits the immediate and anticipated development
2. Code the module action class
3. Code the JSPs
4. Add the module Action Class details and JSPs to cfs-config.xml; now ConcourseSuite will know how to map URL requests to Action Classes and JSPs
5. Add a Main Menu item to cfs-modules.xml, listing ALL action names that have access to the module; now ConcourseSuite will show the menu item in the Main Menu for all actions specified
6. Insert the module permissions and preferences into the [permission_category] and [permission] tables; now the module will be recognized for permissions and configuration in ConcourseSuite
7. Create install and upgrade scripts

ConcourseSuite Framework Model

The ConcourseSuite framework currently utilizes three (3) action class-JSP design models. Model 1 is the simplest and is typical when developing new features. Model 2 consolidates potentially duplicate code at the JSP level by combining form data and re-using the JSP with minor additional logic. Model 3 introduces a Shared Action Class that can be called from other Action Classes when action class code needs to be re-used.

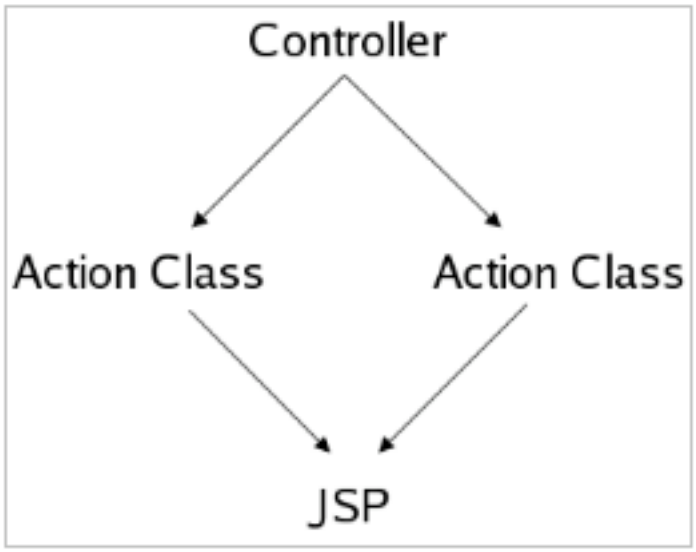
Model 1

The controller executes an Action Class, the Action Class returns a result which calls a standalone JSP. A different JSP is required for showing a list of items, an input form, and a detail page.



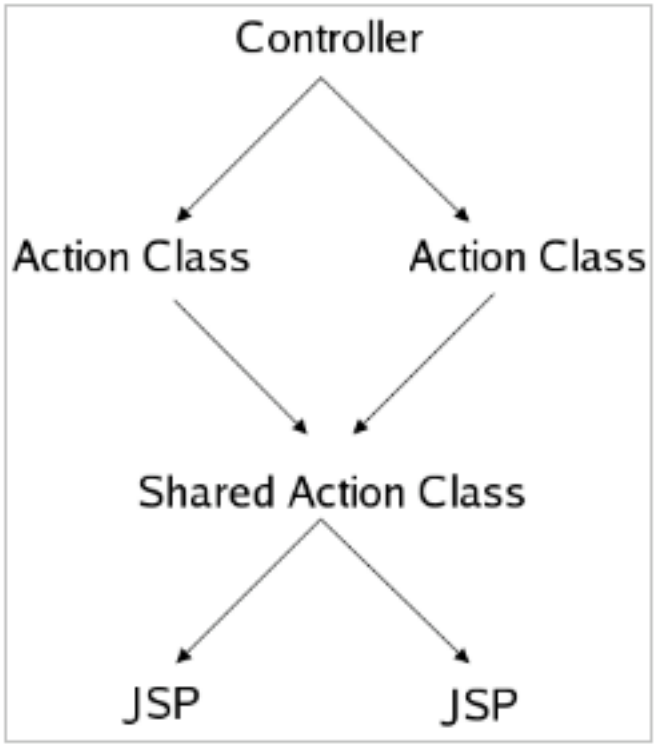
Model 2

The controller executes an Action Class, the Action Class returns a result which calls a shared JSP. Typically a shared JSP is used when presenting an input form that is used for initially inserting data or for modifying data in multiple modules. The JSP may have some basic logic to show the user the word “Save” or the word “Update.” This model adds reusability to Model 1.



Model 3

The controller executes an Action Class, the Action Class requires the generation of Lookup Lists and other form data and returns a result which calls a standalone JSP. Typically this model is used when two completely different modules require the same form to be generated, but in a different user context. In this case, the module specific Action Class is linked to a Shared Action Class, and then returns a module specific JSP.



Setup Site-Based Permissions for objects

Users have site identifiers that identifies the site (a.k.a. division, territory) that they belong to. A user's site restricts the user to manage (i.e., view, add, edit and delete, based on module based permissions) information that belong to the same site that he or she is member of. The site information is stored in several tables in the 'site_id' column in the database and a siteId attribute in the classes. A database record with a null site_id and object with siteId is -1 refer to items for which site information is not recorded. For information on how sites affect user permissions on various objects refer to the section on Territories.

Contact and User (access table) Site Dependency

The access table has a column site_id to store the site that a user belongs to. As you may be aware, all users, and by inference, the records in the access table relate to a contact record. It is mandatory that the value of the site_id in the access table be the same as the value of the site_id in the corresponding contact table. Keeping the the values consistant requires additional maintenance because it reduces the way the tables are normalized, however, this decision has been taken keeping in mind several performance and response time issues.

Contact and Account (organization table) Site Dependency

The Organization table has a column site_id to store the site that an account belongs to. By inference, the records in the database that relate to the organization table belong to the same site as the organization. It is mandatory that the value of the site_id of account contacts in the contact table be the same as the value of the site_id in the corresponding organization table. Keeping the the values consistant requires additional maintenance because it reduces the way the tables are normalized, however, this decision has been taken keeping in mind several performance and response time issues.

Site information in other tables

Several other tables like usergroups, actionplan, etc have a site_id. For e.g., the user_group table stores site_id and this information is used to restrict the membership of the group, the actionplan table has a site_id that restricts whether it can be associated with a specific, ticket, account or lead (contact). If new tables that you may add require that site information be stored, we urge think of two options, (1) if the site information can be inferred from a related organization, contact or access table do not add a site_id column to the table unless performance (i.e., response time due to additional joins) degrades performance, (2) if the table is used to store administrative data (e.g., usergroups,

categories, etc.,) that does not relate to relavant tables that store site information, add the site_id column.

Checking user access permissions on Site related information

User objects are cached in the application and this cached information is compared against all required records using the overloaded isRecordAccessPermitted(...)1, 2 method in the CFSModule.java class. To insure that the validation is secure it is neccessary that the object not be validated against a visible siteId parameter in the URL. This is in addition to UI validations that prevent site related business logic from being violated. This additional check is required as data in ConcourseSuite CRM can be made persistant using means other than the UI, such as the XML-RPC API.

Use Lookup Lists

Lookup lists are used to minimize redundant data in the database; they are also used within an application to limit and display choices to the user. By using the LookupList class, you can easily add new lookup lists to the application cache and user's can modify them using the administrative lookup list editor.

Lookup List Database Table Design

The fields required for a simple lookup list include:

- **code** -- the unique primary key
- **description** -- the value to appear in the lookup list
- **default_item** -- indicates if this value should be selected by default when first displayed
- **level** -- the order in which the value should be displayed
- **enabled** -- indicates whether this value should be displayed to the user or not

Notes:

- The sequence name should be specified, instead of using the database's default
- Sequence names can be up to 31 characters

```
CREATE SEQUENCE lookup_step_actions_code_seq;
CREATE TABLE lookup_step_actions (
  code INTEGER DEFAULT nextval('lookup_step_actions_code_seq') NOT NULL PRIMARY
KEY,
  description VARCHAR(300) NOT NULL,
  default_item BOOLEAN DEFAULT false,
  "level" INTEGER DEFAULT 0,
  enabled BOOLEAN DEFAULT true
);
```

To install your new lookup list into ConcourseSuite CRM, see the section on creating install and upgrade scripts.

Instantiating in an Action Class

Lookup lists can be instantiated and cached by using the systemStatus object to access the list. Once retrieved, the list can be used in the request, but must not be modified because it is shared.

```
// Retrieve Lookup List using the cache
SystemStatus systemStatus = this.getSystemStatus(context);
LookupList list = systemStatus.getLookupList(db, "lookup_step_actions");
// Add the lookup list to the request to be used by a JSP
context.getRequest().setAttribute("stepActionsLookupList", list);
```

A lookup list can also be loaded directly from the database and manipulated before being used in a JSP.

```
LookupList list = new LookupList(db, "lookup_step_actions");
context.getRequest().setAttribute("stepActionsLookupList", list);
```

Accessing from a JSP

With the LookupList object in the request, the method **getHtmlSelect("formFieldName", defaultId)** can be used to render an HTML Select field with the options.

```
<jsp:useBean id="stepActionsLookupList" class="org.aspcfs.utils.web.LookupList"
scope="request"/>
<tr class="containerBody">
  <td class="formLabel">
    <dhv:label name="sales.step.action">Step Action</dhv:label>
  </td>
  <td>
    <%= stepActionsLookupList.getHtmlSelect("stepAction",
otherBean.getStepActionId()) %>
  </td>
</tr>
```

The LookupList object contains both enabled and disabled items, however only the enabled items will be shown. The exception is that if the form object is set to a disabled item, the disabled item will be included in the Lookup List with an **(X)** next to its value to alert the user that a disabled item is being used. The user can then optionally change the value to an enabled value, or leave the existing

disabled value.

Object Validator

ConcourseSuite CRM provides various user-input forms which allow a user to add new content or edit existing data in the CRM. Based on the business rules that govern a particular object, the object should have certain required data associated with it at any point in time. As an example, any Ticket in the system should be associated with a specific Account and it's Contact.

Certain forms are designed to encapsulate a particular object and therefore provide form validation. On form submission, the object represented by the form is auto-populated and fed to the Object Validator, before it can be successfully stored in the database. As the name suggests, the Object Validator performs object validation and verifies if the object satisfies all the rules that it should adhere to, based on the business rules defined for that specific object.

If the object is successfully validated, then it is stored in the database, else the form is returned to the user for review and to re-submit.

com.concursive.crm.web.controller.utils.ObjectValidator;

When a new form that corresponds to a bean is introduced in ConcourseSuite CRM, the action to which the form is posted to, when a user submits it, should call the following method to validate the object.

ObjectValidator.validate(SystemStatus, Connection, Object)

Most of the errors or warnings thrown by a form that corresponds to a bean, are represented by the following constants defined in the object validator. Each constant denotes a particular error/warning and is used to determine the message to be displayed when the form is returned back to the user.

1. REQUIRED_FIELD
2. IS_BEFORE_TODAY
3. PUBLIC_ACCESS_REQUIRED
4. INVALID_DATE
5. INVALID_NUMBER
6. INVALID_EMAIL
7. INVALID_NOT_REQUIRED_DATE

8. INVALID_EMAIL_NOT_REQUIRED

9. INVALID_ENTRY

Register Module Reports

Reports can be installed as part of a new system or as an upgrade to an existing system.

Registering reports for a new system

1. Go into permissions_en_US.xml and locate the category for which you want to install the report under

2. Make sure the category has an attribute: reports="true"

3. At the bottom of the category, you can duplicate an existing report entry, or create a new one from scratch... the format is:

```
<report file="report_file_name.xml" type="user" permission="myhomepage-tasks"
title="Short report title" description="Longer report title"/>
```

4. The permission attribute of report allows a user with the given permission to execute the report; use type="user" as this is not implemented

5. Copy the report file into the source repository, into: src/jasper_reports (using a .xml extension and not .jrxml)

6. Install the database from scratch using "ant installdb" and the new report will appear

Registering reports in an existing system

1. Make sure the steps above are complete as this sets up the environment

2. Run "ant deploy" to copy the report and new code into your CRM

3. Make an upgrade script, based on: src/sql/upgrade/2006-08-29-script02-ananth.bsh; make sure to change the categoryId to the correct module's constant categoryId

4. Execute "ant upgradedb", specify the database and script names, the report will now be registered in an existing system

Adding Portlets

The following steps assume that ConcourseSuite CRM has been deployed and that you want to tightly couple a portlet with the CRM.

The integrated portlets have access to the ConcourseSuite CRM database as well as the CRM application, system, and user objects. This allows the portlets to access key framework items and redirect to crm modules, something that is difficult with jsr-168 alone. This approach enables delivering a single application with embedded portlets that work on the CRM with supported web application servers.

Register the portlet for use with ConcourseSuite CRM

1. Add the contents of your portlet's web.xml file to the main web.xml file. You will see loads of examples of portlet deployment declarations there. (we are working with Tomcat 7 so that this requirement will no longer be necessary in a future CRM release)
2. Add the contents of your portlet's portlet.xml file to the main portlet.xml file. Again you will see many examples of this in the portlet.xml file.
3. If you want your portlet to be user-managable through the web interface, then add an entry for your portlet to icelet_(your language).xml file. You'll see examples in the existing icelet...xml file. This step is not required for typical portlets.
4. Explode your portlet's war file someplace. You'll need to copy every dependency for your portlet to the webapp directory. Following are the steps:
    1. Copy all the JSPs to their proper location under the Tomcat root directory for the web app.
    2. Copy all the libraries that your portlet needs to the WEB-INF/lib directory of the web app. You're on your own at resolving any class loader issues that get introduced by the inclusion of your portlet's libraries in the CRM classpath.
    3. Copy your portlet's classes into the webapps WEB-INF/classes directory. That directory does not exist by default. Your portlet introduces it to the Tomcat class loader mechanism. (or use a .jar and place in WEB-INF/lib which does exist)
5. Startup Tomcat
6. If you registered your portlet for management by an Admin, log in as an administrator and go the to the Admin module where you must add your portlet to a dashboard or some custom tabs for a module.

Integration with Asterisk

ConcourseSuite CRM can make outbound phone calls from an Asterisk extension, and can display screen-pops through an XMPP server for inbound phone calls.

Initial Configuration

1. Setup an Asterisk Server
2. Enable the Asterisk manager configuration entry to allow port listening by a user
3. Configure the CRM with the Asterisk server properties

Sample manager.conf:

```
[general]
enabled = yes
port = 5038
bindaddr = 0.0.0.0
;displayconnects = yes

[crm]
secret = crm
read = system,call,log,verbose,command,agent,user
write = system,call,log,verbose,command,agent,user
```

Inbound Phone Calls

This capability requires the additional configuration of an XMPP Server. When ConcourseSuite CRM is alerted to an incoming phone call by Asterisk, the following steps are taken to alert the user with an instant message; if any of them fail, then no alert is made:

1. ConcourseSuite CRM checks the user list for a valid user with an extension receiving the call
2. ConcourseSuite CRM checks to see if this user has a Jabber instant message address
3. ConcourseSuite CRM looks up the incoming phone number, based on the caller ID details, for a contact that the user has access to
4. ConcourseSuite CRM checks to see if the user is logged into the XMPP server and is listed as available
5. If necessary, the ConcourseSuite CRM Jabber user will request that the target user allow them to be added to their roster

6. ConcourseSuite CRM sends the IM with a link to the contact record

Outbound Phone Calls

The first iteration of this capability simply places a red phone icon next to phone numbers that can be dialed. When the user selects the phone icon

, ConcourseSuite CRM prompts which extension the user is currently at, then proceeds to place a call. The user's phone will ring, then Asterisk will continue placing the call.

Additional dialed calls during the user session will not prompt for the user's extension.

Action Plan Development

Action Plans have many useful capabilities and are being extended throughout. This section is intended to introduce you to the technical side of action plans.

Action Plans

Action Plans are made of a series of steps (activities) and each step can provide a particular type of action that a user can perform while working on that particular step. Action Steps can be further configured to enforce certain behaviour when the user is working on it. Action Plans can be a useful tool in preparing a business plan and tie it with existing objects in ConcourseSuite CRM. Today Action Plans can be tied with 'Accounts' or 'Tickets', but can easily be extended to associate with other objects.

The "User Guide" is a good source of information on how to setup Action Plan Templates as an Administrator, so that Users can work thorough an Action Plan. The following pointers briefly introduce the way Action Plans work

- An Action Plan is made of one or more Action Phases
- An Action Phase has one or more Action Steps
- An Action Plan Template can be setup by the Admin and is associated with a particular module For e.g. 'Accounts'
- Every Account can now have one or more Account Action Plans
- When an Account Action Plan is created, a copy of the Action Plan Template is created and linked with the Account for the user to work through the plan
- An Action Step can be configured during setup to specify its behaviour (type of action, required etc) and ultimately the way the user would interact with the Step.

Action Plan Database Schema

The Action Plan schema comprises of the following main database relations:

- **action_plan** (ActionPlan) -- represents an Action Plan Template
- **action_phase** (ActionPhase) -- represents an Action Phase Template
- **action_step** (ActionStep) -- represents an Action Step Template

- **action_plan_work** (ActionPlanWork) -- copy of the Action Plan Template for the user to work with
- **action_phase_work** (ActionPhaseWork) -- copy of the Action Phase Template for the user to work with

- **action_item_work** (ActionItemWork) -- copy of the Action Step Template for the user to work with

Action Plan Module Constants

There are several constants that are defined in Action Plans (ActionPlan) that allow the developer to associate the plan with certain modules and to enforce certain behaviour. All of the Action Plan Module Constants are available in the the database relation **action_plan_constants.** Each Constant refers to a particular module.

- **action_plan_editor_lookup** refers to a particular constant that makes the Editor available under that module
- **action_plan** refers to a particular constant that makes the Action Plan available under that module
- **step_action_map** refers to a particular constant that makes this Step Action available under the module's Action Plan

Action Plan UI

The Action Plan User Inteface is made of various sections with varying complexity. Several JSPs & Custom Taglib Handlers are used to deliver the content and user interaction. Each section is briefly explained below.

- **Phase Indicator** section displays all the phases that make up the plan in a row format. The current phase in the plan is highlighted.
- **Plan Details** displays information about the Plan itself. Information displayed here includes Manager, Assignee, Prospect Name, Date the plan became active on etc.
- **Plan Activities** section displays each Phase and all the steps that make up the phase. The current phase and the current step that the user needs to work on is highlighted. User interacts with the plan by checking off a step if it is complete. Each step might have a particular action that the user needs to do before he can mark a step as complete. The JSP ensures that the current step in the plan is written out in such a way that appropriate

Additional items:

- Developing Action Plan Steps
- Building Action Plan Reports

Adding Action Plan Support to Modules

Action Plans are made of a series of steps (activities) and each step can provide a particular type of action that a user can perform while working on that particular step. Action Steps can be further configured to enforce certain behaviour when the user is working on it. Action Plans can be a useful tool in preparing a business plan and tie it with existing objects in ConcourseSuite CRM. Today Action Plans can be tied with 'Accounts' or 'Tickets', but can easily be extended to associate with other objects. For more technical information on Action Plans, see Action Plan Development.

Here we describe how one can add Action Plan support for Leads module. A developer needs to perform the following changes to support Action Plans.

- Admin support for Leads Action Plans
- Action Plan support in the Leads module

Admin Module: Leads Action Plans

When the Admin navigates to ‘Configure Modules > Leads’ a new configuration item ‘Action Plan Editor’ should be displayed.

When the admin clicks on ‘Action Plan Editor’ the ‘Leads > Action Plan Editors’ page needs to be displayed. The admin can edit the action plans that are associated with Leads. Then the admin can proceed to add and configure new Action Plans and review existing ones similar to ‘Admin > Configure Modules > Accounts > Action Plan Editors > Action Plans’

- Add a new permission sales-leads-action-plans for Lead Action Plans with VAED attributes. Add the permission under “Leads” in permissions_en_US.xml. Provide a BSH upgrade script that will insert this permission into the database.
- Set the ‘Leads’ module to have action plans by providing ‘actionPlans="true"’ attribute for category ‘Leads’ in permissions_en_US.xml. Provide a BSH script which will enable existing systems to have Lead Action Plans.
- Specify a new Constant in ActionPlan.java for LEADS = 711071244 as ‘leads’. Add a <record> element for ‘action_plan_constants’ in lookuplists_en_US.xml with ‘leads’ data. Provide a BSH script similar to 2005-09-12-script01-partha.bsh to add this record to an existing system based on the dictionary available.
- When the user is working through an action plan, each step in the plan can have a particular type of user action tied to it. There is a predefined pool of available user actions that can be used while configuring an action step. The developer needs to determine the subset of these actions that are visible to the admin while configuring Lead Action Plans & Steps. Provide a BSH script which defines the various actions

available while configuring Action Steps for Lead Action Plans. Refer to 2005-09-16-script01-partha.bsh. The Lead step actions will be the subset that is available for Account Action Plans. So the section under Accounts Mappings in the bsh should be provided.

- The LEADS constant must be used for action_plan.link_object_id when adding action plan templates for Leads in the Admin section. When adding action plans for a particular Lead, the action_plan_work.link_module_id and action_item_work.link_module_id should be set using the LEADS constant.

Leads Module: Action Plan Support

When the user navigates to the Leads module and selects a particular Lead object to review its details, based on the user's role there are several side-tabs that are visible. If 'Action Plans' are visible to the user he can add any number of action plans based on the action plan templates that have been configured by the admin.

- Include the ‘sales-leads-action-plans’ permission for ‘actionPlans’ submenu under ‘leads’ container in cfs-container_menus.xml.
- Provide the action commands ‘View’, ‘Add’, ‘Insert’ and ‘Details’ in SalesActionPlans.java. Please refer to AccountActionPlans.java to provide similar action commands. Use action command chaining to use common functionality in ActionPlans.java.
- The Leads Action Plan list page should display a drop-down menu for each record with the following actions:
    - View Action Plan
    - Reassign
    - Review Notes
    - Archive
    - Delete

Handling ConcourseSuite Data

Users interested in ConcourseSuite CRM are companies that already have a CRM application with lots of valuable Customer data or are companies that have Customer data in various other forms (Excel documents, Custom database etc) and would like to use ConcourseSuite CRM going forward.

To bring in Customer data from various sources and also to move data out of ConcourseSuite, users can leverage some of the existing tools & features which allow the user to import/export data in and out of a ConcourseSuite database.

Contact/Account Importers

The following 3 importers are available for bringing in people and organization information into ConcourseSuite.

- **Leads Importer** - Allows importing of people as Leads into ConcourseSuite CRM (*Available in the Leads module)*
- **Contacts Importer** - Allows importing of people as General Contacts into ConcourseSuite CRM (*Available in the Contacts module)*
- **Account Contacts Importer** - Allows importing of people as Account Contacts into ConcourseSuite CRM (*Available in the Contacts module)*

```
Note: When importing Contacts using the Account Contacts Importer, if the Company
Name is present in the record being
imported, then an Account is created and the Contact is associated with the new
Account that was created. If the Company
Name is NOT present, then the Contact is imported as an Individual Account and you
can see an Account as well as a Primary
Contact record in ConcourseSuite.
```

Data Transfer Application

ConcourseSuite CRM has a Data Transfer Application (Reader/Writer) written in Java which can read data from various input formats and write data to various output formats. These applications can interface with files, databases, web services, etc.

To import related data, for example Accounts, Contacts, Opportunities, and Notes, a Reader can be developed and executed.

ConcourseSuite CRM includes a Writer that knows how to write data into ConcourseSuite CRM, and comes with some basic Readers.

Readers are Java programs that send data to a Writer and associates data during the reading/writing process. The reader and writer are configured and executed by the Transfer application.

Some of the Readers available in ConcourseSuite are:

- ImportAccountContacts - Reads from a .CSV file and populates Organization & Contact objects
- ImportAccounts - Reads from a .CSV file and populates Organization objects
- ImportGeneralContacts - Reads from a .CSV file and populates Contact objects
- CFSDatabaseReader - Reads data out of a ConcourseSuite database

Some of the Writers available in ConcourseSuite are:

- TextWriter - Writes to a text file
- Cfshttpxmlwriter - Sends XML packets to a remote ConcourseSuite server (using ConcourseSuite's XML API)

A configuration file of the form shown below is to be used to execute the Transfer Application. The config file needed by the Transfer application, has information about the Reader that needs to be loaded and a Writer that needs to be passed to the Reader to read/write data.

```
<data-import-config>
  <description>
    Copies contacts from an Excel generated CSV file into ConcourseSuite under a
new Electronic Import user
  </description>
  <reader class="org.aspcfs.apps.transfer.reader.cfs.ImportAccountContacts">
    <propertyFile>@PROPERTY.FILE@</propertyFile>
    <csvFile>@CSV.FILE@</csvFile>
  </reader>
  <writer
class="org.aspcfs.apps.transfer.writer.cfshttpxmlwriter.CFSHttpXMLWriter">
    <url>http://@URL@/ProcessPacket.do</url>
    <id>@ID@</id>
    <code>@CODE@
```

```
  <systemId>@SYSTEM.ID@</systemId>
</writer>
```

</data-import-config>

</code>

The Readers and Writers exchange Data using a DataRecord object which encapsulates a database record in ConcourseSuite.

Backup & Restore

The Backup & Restore application is built on top of the Data Transfer Application explained above. There a set of Readers and Writers which read from a ConcourseSuite database and store the data is an XML format in a file and then read the data from the XML file and store it in the database using a specialized writer.

Backup Process

The Backup tool is comprised of a Reader/Writer combination and uses the following in ConcourseSuite CRM

```
CFSDatabaseReader
CFSXMLWriter
```

The CFSDatabaseReader reads all of the database records from any ConcourseSuite CRM database and passes it to a Writer. The CFSXMLWriter is used in the backup process to write the records into an XML file which conforms to a particular Centric Backup File Format.

Restore Process

The Restore tool is comprised of a Reader/Writer combination and uses the following in ConcourseSuite CRM

```
CFSXMLReader
CFSXMLDatabaseWriter
```

The CFSXMLReader reads all of the records from the backup XML file and passes it on to the CFSXMLDatabaseWriter which stores the records to the database.

Developer Notes

The Backup and Restore application needs to be maintained. The following is a list of items that need to be maintained:

- As new tables and new columns are introduced into the database schema the import-mappings.xml file needs to be updated with mappings to reflect the new columns and new tables
- As new tables are introduced, the existing data reader classes (eg: ImportCommunications.java) need to be updated to read from these new tables.
- If new modules are introduced in CCRM, they might require a new data reader class that reads from all the database tables that were introduced for the new module in the right sequence. The import-mappings.xml should also be updated with the new data reader information.
- The restore application will need to be tested to see if it works with the new modules/tables. The Database Writer used by the restore process might need to be upgraded based on any required functionality

Centric Backup File Format

Here is a sample record that is output by the CFSXMLWriter. The following XML will be parsed by a Database Writer to populate the org.aspcfs.modules.admin.User object and populate it via reflection and store it in the database

```
<dataRecord action="insert" name="user" shareKey="false">
  <dataField alias="guid" name="id">1</dataField>
  <dataField name="username">ananth</dataField>
  <dataField name="password">792fef441691aa41135a15c1478a5ee4</dataField>
  <dataField name="encryptedPassword">792fef441691aa41135a15c1478a5ee4</dataField>
  <dataField name="lastLogin">2007-06-22 14:28:02.324</dataField>
  <dataField name="ip">0:0:0:0:0:0:0:1%0</dataField>
  <dataField name="timeZone">America/New_York</dataField>
  <dataField name="startofday"></dataField>
  <dataField name="endOfDay"></dataField>
  <dataField name="expires"></dataField>
  <dataField name="enabled">true</dataField>
  <dataField name="entered">2007-03-29 16:27:00.396</dataField>
  <dataField name="modified">2007-03-29 16:27:00.396</dataField>
  <dataField name="currency">USD</dataField>
  <dataField name="language">en_US</dataField>
  <dataField name="webdavPassword">e8613b3e0d8f1581eec4a5fa17fac7ff</dataField>
  <dataField name="hidden">false</dataField>
  <dataField name="hasWebdavAccess">false</dataField>
  <dataField name="hasHttpApiAccess">false</dataField>
  <dataField name="addContact">false</dataField>
</dataRecord>
```

Using Web Services

As of Version 6, ConcourseSuite no longer maintains Web Services using SOAP. The preferred method is the HTTP-XML API.

If you intend to use them, these methods may or may not work. ConcourseSuite provides a set of classes that can be exposed as Web Services, and

can be consumed by external applications using SOAP. To enable Web Services

the following software needs to be installed as a separate webapp under you

tomcat installation.

Apache Axis 1.2

http://ws.apache.org/axis/

You must have AXIS_HOME property defined as an environment variable OR as a

property in a file called "home.properties"

$ export AXIS_HOME=/path/to/apache-tomcat/webapps/axis

% Using System, in Windows Control Panel, set:

AXIS_HOME=c:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\axis

Make sure to edit 'Axis' related properties in the following properties file

```
$CENTRIC_HOME/WEB-INF/build.properties

AXIS.WEBAPP=/axis
AXIS.HOST=127.0.0.1
AXIS.PORT=8080
```

To make the web service classes available to external applications, register

ConcourseSuite's Web Services with Axis. Run the following ant command:

```
  $ ant ws

  The <axis-admin> [[http://ws.apache.org/axis/java/ant/axis-admin.html task]] is
used to register CCRM services.
```

By default, Axis permissions require that you execute this command on the web server itself using localhost or 127.0.0.1.

If the command is successful, then you should be able to point your browser at

the following url and you will see the ConcourseSuite's Web Services that were deployed

http://{yourhostname}/{axis.webapp}/servlet/AxisServlet/

Using ConcourseSuite Outlook Plugin

- Outlook Plugin FAQ

Outlook Plugin FAQ

Adding support for a new database

For database independence, we use JDBC and SQL standards compliance. We've taken a simple, yet efficient approach in which all of our objects know how to talk to the database directly, using a connection pool and DatabaseUtils Class for database-specific syntax. We don't use triggers, views, BLOBs, cursors or stored procedures.

In general, our Base and List Classes use:

- SELECT with LEFT JOINs
- SELECT with WHERE clause uses IN (SELECT)
- SELECT with OFFSET and LIMIT, or TOP to retrieve paged list of records (db dependent)
- SELECT with EXISTS
- INT, VARCHAR(300+), TEXT, FLOAT, TIMESTAMP or DATETIME types
- TIMESTAMPS with NULL values and without auto-timestamping behavior
- INSERT with retrieval of SEQUENCE or AUTONUMBER before or after insert (db dependent)
- TRANSACTIONS with COMMIT and ROLLBACK
- CURRENT_TIMESTAMP
- Cast functions as defined in DatabaseUtils (db dependent)

Changes that need to be made in ConcourseSuite CRM

1. Verify that the database meets the application's implemented capabilities; are there any limits on table names, field names, datatype lengths, indexes, or sequences? If so, an algorithm will be needed to truncate the default table, field, index, and sequence names
2. Copy the JDBC driver into ConcourseSuite CRM's lib directory, the driver name must include version information
3. Implement the database creation scripts based on the included scripts; PostgreSQL is the reference implementation, however the database may be closely related to another database; confirm the characteristics of the database first
4. Change ConcourseSuite CRM's master.properties file to include a connectivity example
5. Change DatabaseUtils.java to support the new database
6. Change PagedListInfo.java to support the new database
7. Change CusomFieldRecord.java to support the new database
8. Change Setup.java and configure_database.jsp for application installation

Testing

1. First configure ConcourseSuite CRM as usual, but specify the new database in build.properties during deployment
2. Manually create the database
3. Run "ant installdb" to create the database schema and install the base crm data
4. Repeat until installdb works
5. Try to login and test the application

Exercises

The following exercises introduce the ConcourseSuite CRM framework and are intended to be completed in order:

- Add a sub-menu to an existing module
- Add a form page to a sub-menu item
- Register an editable lookup list in the Admin module
- Write a business process that sends an email when a form is filled out

Appendix A: Cloning a module

The following steps list how the Contacts module can be cloned to another module, e.g., Personnel module

This is a first effort to list the steps. Suggestions and Modifications are encouraged.

- Copy the action class from /src/main/java/com/concursive/crm/web/modules/contacts/actions/ExternalContacts.java

to /src/main/java/com/concursive/crm/web/modules/personnel/actions

- Change the package information for the action classes in

/src/main/java/com/concursive/crm/web/modules/personnel/actions

from

package com.concursive.crm.web.modules.contacts.actions;

to

package com.concursive.crm.web.modules.personnel.actions;

```
    <menu>
      <action name="ExternalContacts"/>
      <action name="ExternalContactsOpps"/>
      <action name="ExternalContactsOppComponents"/>
      <action name="ExternalContactsCalls"/>
      <action name="ExternalContactsCallsForward"/>
      <action name="ExternalContactsPrototype"/>
      <action name="ExternalContactsImports"/>
      <action name="ExternalContactsHistory"/>
      <action name="ExternalContactsMessages"/>
      <page title="Contacts"/>
   ....
  </menu>
```

- In cfs-modules.xml copy the section

for the contacts module and paste it below the closing menu tag (or where ever else you want to position it.) Rename the title to "Personnel"

- Create permissions in the database to control access to personal module (refer to development document to update permissions in an existing installation and to add them during the installation process)
- Replace the permissions in cfs-modules for the personnel module.

```
<!-- Submenu used in the Contacts module -->
```

- The cfs-container_menus.xml draws the subtabs for each record. Search for
    - They would contain the definition of the sub tabs for the contacts module. Include the permissions for the personnel module seperate with comma for the value attribute of the permission tag (need to verify if this works)
- Copy all the jsps in web/jsp/contacts to web/jsp/personnel. Right now, I perceive this is require so that the forms submit to the correct action classes. An alternative suggestion is welcome.
    - One alternative I percieve is to have a _include.jsp for each page in the contact/ directory and have the wrapper buttons in the personnel directory.
- In cfs-config.xml, define what pages each of the methods of the action classes in the personnel module forward to.

E.g.,

```
 <action name="ExternalContacts"
        class="com.concursive.crm.web.modules.contacts.actions.ExternalContacts">
....
       </action>
```

needs to be copied and changed to

E.g.,

```
   <action name="Personnel"
class="com.concursive.crm.web.modules.personnel.actions.ExternalContacts">
 ....
        </action>
```

Change file path of jsps from contacts to personnel.

This needs to be repeated for each of the action classes that are now in the personnel module.

Automated Configuration Without Human Intervention

Q: What are the possibilities to perform an automated configuration, without human interaction?

A: There is a mechanism for ConcourseSuite CRM to locate the preferences during startup. If the application finds the necessary files and configuration, then it skips the web-based installer. While the .war wasn't meant for this, it could be tweaked for the specific case you mention.

There are 4 components:

1) ConcourseSuite CRM must know where the external fileLibrary is located. This can be done in one of two ways...

A) The Java preferences file must be installed or setup for the corresponding O/S. In the case of WAS, a /WEB-INF/instance.property file must be located in the ConcourseSuite CRM .war. This is a unique identifier if multiple ConcourseSuite CRM .wars are located on the same system. This is used to map the .war to a fileLibrary using the Java Preferences API. This is generally done using the web-based installer.

or

B) ConcourseSuite CRM looks for a path.txt file in a specific place depending on the O/S. If it finds it, then it records the path using the Java Preferences API.

2) Once ConcourseSuite CRM knows where the fileLibrary path is, it checks for a build.properties file there; this file declares all of the preferences for ConcourseSuite CRM including Connection URL and more. The version of ConcourseSuite CRM must be accurately recorded here or else the webapp goes into upgrade mode.

3) The fileLibrary must also have a default set of files. One of these files is a unique license for the system. Typically this is generated during the web-based install and requires internet access. We could work something out to pre-generate these. The other files are easy to copy from a working instance.

4) The ConcourseSuite CRM database must exist and have the schema and default data.

Here are the details that should allow you to install ConcourseSuite CRM and bypass the web-based ui...

1. Create a text file in one of the following locations... (Linux is specified first and /opt is tried if it exists)

/opt/centric_crm/fileLibrary/path.txt

/var/lib/centric_crm/fileLibrary/path.txt

c:\CentricCRM\fileLibrary\path.txt

/Library/Application Support/CentricCRM/fileLibrary/path.txt

2. This text file can have 2 lines maximum, the first line being an optional comment:

1. This file is used by ConcourseSuite CRM to determine where the fileLibrary is

/opt/centric_crm/fileLibrary/instance1

Uninstalling

There are several components to a ConcourseSuite install...

1. The web application itself (crm.war) as well as the 'crm' directory in the 'webapps' directory.
2. The fileLibrary, chosen and created during installation
3. A Java Preference determined by the OS and created during installation

Cleaning up those three items, completely removes ConcourseSuite.

Default File Library Paths	
Linux	/var/lib/concursive/crm/fileLibrary/
Mac OSX	/Library/Application Support/Concursive/crm/fileLibrary/
Unix	/opt/concursive/crm/fileLibrary/
Windows	c:\Concursive\crm\fileLibrary\

Default Java Preference Locations	
Linux	/root/.java/.userPrefs/com/concursive/crm/
Mac OSX	~/Library/Preferences/com.concursive.crm.plist
Windows	Windows registry entry (search for concursive)