Sugar Developer Guide 6.7

Sugar Developer Guide 6.7

The Sugar Developer Guide is designed for developers who are new to Sugar, or to CRM and Web-based applications. This guide introduces you to some basic CRM concepts and helps you get familiar with the Sugar system. It describes how to configure and customize the Sugar platform for a broad range of tasks applicable to companies, government agencies and other organizations that have a need to manage business relationships with people.

Readers are expected to have basic programming and software development knowledge, be familiar with the PHP programming language and the SQL database language.

Please note that this developer guide is for the 6.7.x release of Sugar which is only supported on On-Demand.

Last Modified: 11/18/2015 02:08am

Preface

Overview

Welcome to Sugar, an open source Customer Relationship Management (CRM) application. Sugar enables organizations to efficiently organize, populate, and maintain information on all aspects of their customer relationships.

The Sugar Application

Sugar provides integrated management of corporate information on customer accounts and contacts, sales leads and opportunities, plus activities such as calls, meetings, and assigned tasks. The system seamlessly blends all of the functionality required to manage information on many aspects of your business into an intuitive and user-friendly graphical interface.

Sugar also provides a graphical dashboard to track the sales pipeline, the most successful lead sources, and the month-by-month outcomes for opportunities in the pipeline.

The Sugar Community

Sugar is based on an open source project and therefore advances quickly through the development and contribution of new features by its supporting community.

Welcome to the community!

Audience

The Sugar Developer Guide provides information for developers who want to extend and customize Sugar functionality using the customization tools and APIs provided in the Sugar Ultimate, Enterprise, Corporate, Professional, and Community Editions.

Application Overview

Sugar consists of modules which represent a specific functional aspect of CRM such as accounts, various activities, leads, and opportunities. For example, the Accounts module enables you to create and manage customer accounts while the Meetings, Tasks, Emails, and Notes modules enable you to create and manage activities related to accounts, opportunities, etc. Sugar modules are designed to help you manage your customer relationships through each step of their life cycle, starting with generating and qualifying leads, through the selling process, and on to customer support and resolving reported product or service issues. Since many of these steps are interrelated, each module displays related information. For example, when you view the details of a particular account, the system also displays the related contacts, activities, opportunities, and bugs. You can view and edit this information and also create new information.
As a developer, Sugar gives you the ability to customize and extend functionality within the base CRM modules. You can customize the look and feel of Sugar across your organization, or you can create altogether new modules as well as build entirely new application functionality to extend these new modules.

Core Features

Sales Force Automation

- Lead, contact, and opportunity management to pursue new business, share sales information, track deal progress, and record deal-related interactions
- Account management capabilities to provide a single view of customers across products, geographies, and statuses

- Automated quote and contract management functionality to generate accurate quotes with support for multiple line items, currencies, and tax codes
- Sales forecasting and pipeline analysis to give sales representatives and managers the ability to generate accurate forecasts based on sales data in Sugar
- Sugar dashboards to provide real-time information about leads, opportunities, and accounts
- Sugar plug-ins for Microsoft Office to integrate your CRM data with Microsoft's leading productivity tools
- Sugar mobile applications for iPhone and Android to access contacts, opportunities, accounts, and appointments in commercial editions of Sugar while logging calls and updating customer accounts
- Sugar mobile browser to access mobile functionality through any standards-based web browser

Marketing Automation

- Lead management for tracking and cultivating new leads
- Email marketing for touching prospects and customers with relevant offers
- Campaign management for tracking campaigns across multiple channels
- Campaign reporting to analyze the effectiveness of marketing activities
- Web-to-Lead forms to directly import campaign responses into Sugar to capture leads

Customer Support

- Case management to centralize the service history of your customers, and monitor how cases are handled
- Bug tracking to identify, prioritize, and resolve customer issues
- Customer self-service portal to enable organizations to provide self-service capabilities to customers and prospects for key marketing, sales, and support activities
- Knowledge Base to help organizations manage and share structured and unstructured information

Collaboration

- Shared email and calendar with integration to Microsoft Outlook
- Activity management for emails, tasks, calls, and meetings
- Content syndication to consolidate third-party information sources
- Sugar mobile functionality for wireless and PDA access for employees to work

when they are away from the office

Reporting

- Reporting across all Sugar modules
- Real-time updates based on existing reports
- Customizable dashboards to show only the most important information

Administration

- Edit user settings, views, and layouts quickly in a single location
- Define how information flows through Sugar (workflow management) and the actions users can take with information (access control)
- Customize the application in Studio to meet the exact needs of your organization
- Create custom modules in Module Builder to develop the additions to the application that are unique to your business to ensure all the critical interactions can be easily tracked and analyzed.
- Implement Sugar Logic formulas and dependencies that help define the workflow and automated calculations to guide your users efficiently through their daily responsibilities.

Last Modified: 11/18/2015 01:08am

Introduction

Overview

Sugar was originally written on the LAMP stack (Linux, Apache, MySQL and PHP). Since version 1.0, the Sugar development team has added support for every operating system (including Windows, Unix, and Mac OSX) on which the PHP programming language runs for the Microsoft IIS web server and the Microsoft SQL Server, IBM DB2, and Oracle databases. Designed as the most modern web-based CRM platform available today, Sugar has quickly become the business application standard for companies around the world. See the Supported Platforms page for detailed information on supported software versions and recommended stacks.

Background

Sugar is available in five editions: the Community Edition, which is freely available for download under the GPLv3 public license, and the Professional, Corporate, Enterprise, and Ultimate editions, which are sold under a commercial subscription agreement. All five editions are developed by the same development team using the same source tree with extra modules available in the Professional, Corporate, Enterprise, and Ultimate editions. Sugar customers using the Professional, Corporate, Enterprise, and Ultimate editions also have access to Sugar Support, Training, and Professional Services offerings. Contributions are accepted from the Sugar Community, but not all contributions are included because SugarCRM maintains high standards for code quality.

From the very beginning of the SugarCRM Open Source project in 2004, the SugarCRM development team designed the application source code to be examined and modified by developers. The Sugar application framework has a very sophisticated extension model built into it allowing developers to make significant customizations to the application in an upgrade-safe and modular manner. It is easy to modify one of the core files in the distribution; you should always check for an upgrade-safe way to make your changes. Educating developers on how to make upgrade-safe customizations is one of the key goals of this Developer Guide.

6 Basic Development Rules for Sugar Products

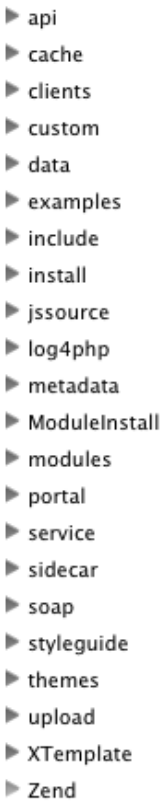
Unless SugarCRM has given you express permission to do so, the following are what not to do when you are configuring, customizing or modifying this Sugar product:

1. Do not remove or alter any SugarCRM or Sugar copyright, trademark or proprietary notices that appear in the Sugar products.
2. Do not "fork" the Sugar software (e.g., take a copy of source code from this product and start independent development on it, creating a distinct and separate piece of software).
3. Do not modify, remove or disable any portion of SugarCRM's "Critical Control Software."
4. Do not combine or use the Sugar products with any code that is licensed under a prohibited license (e.g., AGPL, GPL v3, Creative Commons or another similar license that would "taint" the Sugar products and require you to share the source code for this product with a third party).
5. Do not use any part of the Sugar products for the purpose of building a competitive product or service or copying its features or user interface.
6. Do not use the Sugar products to develop or enhance SugarCRM's "Sugar Community Edition" product or any software code made to work with that open source product.

Application Framework

The Sugar application code is based on a modular framework with secure entry points into the application (e.g. index.php or soap.php). All modules, core or custom, must exist in the <sugar root>/modules/ folder. Modules represent business entities or objects in Sugar such as Contacts, and the object has fields or attributes that are stored in the database, as well as a user interface (UI) for the user to create and modify records. A module encompasses definitions for the data schema, user interface, and application functionality.

The structure of Sugar's root directory is shown below.



Directory Structure

SugarCRM code resides in various directories within the Sugar installation. The structure of the directories within the Sugar application consists of the following root level directories:

- api : v6+ REST API.
- cache : Various cache files written to the file system to minimize database accesses and store user interface templates created from metadata

- clients : Files needed for the new view frameworks in 6.6+
- custom : Location for upgrade-safe customizations such as custom field definitions, user interface layouts, and business logic hooks
- data : Key system files, most notably the SugarBean base class which controls the default application logic for all business objects in Sugar
- examples : Examples of how to extend Sugar, never run from the main app and should be removed from production instances
- include : Many Sugar utility functions as well as other libraries that Sugar utilizes as part of its operations are located here. Most notable in this directory is the utils.php file that contains the most widely used utility functions.
- jssource : Non-minified versions of JS files
- log4php : This has been deprecated as it has been replaced by LoggerManager.
- metadata : Relationship metadata for all many-to-many data relationships between the business objects
- ModuleInstall : Code used to support the Sugar Module Loader
- modules : Contains all modules in the system. Custom modules installed through the Module Loader exist here as well.
- portal : Customer portal code v1
- portal2 : Customer portal code v2 for 6.6+ (Enterprise and Ultimate only)
- service : Web Services code for SOAP and REST v2 - v4_1
- sidecar : Core files used for the sidecar framework.
- soap : Web Services code for SOAP v1; this has been deprecated.
- styleguide : Style guide that the entire application adheres to for a consistent presentation
- themes : Files used to display the various themes
- upload : Files uploaded into the application such as Note Attachments or Documents reside in this directory (refer to upload_dir parameter in the config.php file)
- XTemplate : Old templating library that is being phased out and replaced by Smarty, but some code still may use it.
- Zend : Classes used in the framework

Key Concepts

These are the main files, classes and application concepts that comprise the Sugar platform.

Application Concepts

- Controller : Directs all incoming page requests. This can be overridden in each module to change the default behavior and relies on Entry point

parameters (described below) to serve the appropriate page.
- Views : A set of user interface actions managed by the Controller, the default views in Sugar include the detail view, edit view, and list view.
- Display strings : Sugar is fully internationalized and localizable. Every language pack has its own set of display strings which is the basis of language localization. There are two types of display strings in the Sugar application: application strings and module strings. Application strings contain the user interface labels displayed globally throughout the application. The $GLOBALS['app_strings'] array contains these labels. The $GLOBALS['app_list_strings'] array contains the system-wide drop-down list values. Each language has its own application strings variables. The $GLOBALS['mod_strings'] array contains strings specific to the current, or in-focus, module.
- Dropdown lists : Dropdown lists are represented as name => value array pairs located in the application strings mentioned above. The name value is stored in the database where the value is displayed in the Sugar User Interface (UI). Sugar enables you to create and edit drop-down lists and their values through the UI in Studio. Use the handy get_select_options_with_id() utility function to help render the <select> input options to work with drop-down lists in edit views. Use the translate() utility function to translate the string key you are working with into the user's currently selected display language.

Files

- SugarBean.php : This file located under the <sugar root>/data folder contains the SugarBean base class used by all business entity or module objects. Any module that reads, writes, or displays data will extend this class. The SugarBean performs all of the heavy lifting for data interactions, relationship handling, etc.
- modules.php : The modules.php file contains several variables that define which modules are active and usable in the application.

Variables

- $dictionary : The $dictionary array contains all module field variable definitions (vardefs), as well as the relationship metadata for all tables in the database. This array is dynamically built based upon the vardefs.php definitions.

Entry Points

The primary user interface entry point for Sugar is through index.php located in the root Sugar folder. The main parameters for most calls are:

- module : The module to be accessed as part of the call
- action : The action within the module
- record : The record ID

The following is a sample URL for a Sugar call:

```
http://{sugar_url}/index.php?module=Contacts&action=DetailView&record=
d545d1dd-0cb2-d614-3430-45df72473cfb
```

This URL invokes the detail view action from within the Contacts module to display the record denoted by the record request value.

Other commonly used parameters are return_module , return_action, and return_id. This group of request parameters is used when a user cancels an action such as creating or editing a record.

Note: As of Sugar 5.1, entry points were consolidated into index.php. Previous versions had other files as entry points into the application.

Module Framework

All modules, out-of-the-box (OOTB) or custom, are placed in the <sugar root>/modules/folder. Modules are created to represent an object (such as Contacts) in Sugar, store the object's data points in the database, and provide a UI to create, edit, and delete object records.

The Application Framework section previously mentions an example of a typical call for a detail view action within a module. There are five main actions for a module:

- List View : This Controller action enables the search form and search results for a module. Users can perform actions such as delete, export, update multiple records (mass update), and drill into a specific record to view and edit the details. Users can see this view by default when they click one of the module tabs at the top of the page. Files in each module describe the contents of the list and search view.
- Detail View : A detail view displays a read-only view of a particular record. Usually, this is accessed via the list view. The detail view displays the details of the object itself and related items (subpanels).
  Subpanels are miniature list views of items that are related to the parent

object. For example, tasks assigned to a project, or contacts for an opportunity will appear in subpanels in the project or opportunity detail view. ./<module>/metadata/detailviewdefs.php defines a module's detail view layout. ./<module>/metadata/subpaneldefs.php defines the subpanels that are displayed in the module's detail view page.
- Edit View : The edit view page is accessed when a user creates a new record or edits details of an existing one. edit view can also be accessed directly from the list view. <module>/metadata/editviewdefs.php defines a module's edit view page layout.
- Save : This Controller action is processed when the user clicks Save in the record's edit view.
- Delete : This action is processed when the user clicks Delete in the detail view of a record or in the detail view of a record listed in a subpanel.

The following are driven by the UI framework. This framework relies on metadata files in the requested module.

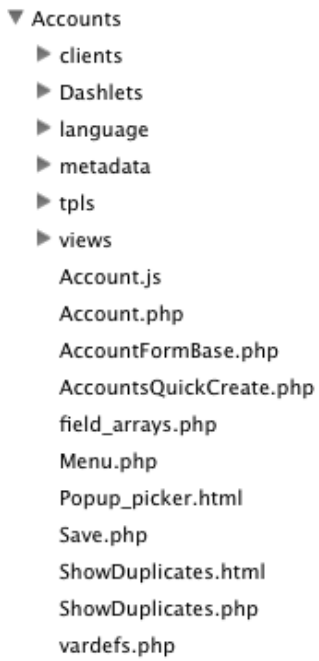
- ./<module>/metadata/listviewdefs.php describes the layout of the list view.
- ./<module>/metadata/searchdefs.php describes the search form tabs above the list view.
- ./<module>/metadata/editviewdefs.php describes the layout of the edit view.
- ./<module>/metadata/detailviewdefs.phpdescribes the layout of the detail view.

In addition to the action files described above, the following files are located in the ./<module>/ folder:

- forms.php : This file contains two functions to render specific JavaScript for validation or other actions during edits/saves. By default you can leave these empty and have them return '';
- Menu.php : This file is responsible for rendering the Shortcuts menu, which was renamed as Actions menu as of Sugar 6.0. In Community Edition, the Actions menu displays below the module tabs and the Last Viewed bar. In Sugar Ultimate, Enterprise, Corporate, and Professional editions, the Actions menu displays on every module tab. By default, you usually add a link to create a new record, and a link to the list view to search.
- Popup.php : This file acts as a wrapper to the central Popup class located under the utils folder. It is called when a module wants to get a popup list of records from a related module. The central Popup class uses the Popup_picker.html and ./<module>/metadata/popupdefs.php file to render the popup.
- Popup_picker.html : This is used by the central Popup class to display a module's pop-ups.
- vardefs.php : The vardefs.php metadata file defines db and non-db fields for

Sugar objects as well as relationships between objects.
- field_arrays.php : This file is deprecated as of Sugar version 5.1. It has been phased out over time with the addition of metadata structures in the application, most notably the vardefs metadata.

The following image displays the subfolders within a module folder:



The following sub-folders are located in the ./<module>/ folder:

- Dashlets : These are drag-and-drop forms displayed on the Sugar Home and Dashboard tabs. Sugar Dashlets display any data (including data from external connectors), and list view and Chart data for the modules. As a developer of a custom module, you can create a Sugar Dashlet for your new module. For each Sugar Dashlet you create, you will place the necessary files in the ./<module>/Dashlets/ folder.
- language : This folder holds the strings files for the module. By default you will have an en_us.lang.php file containing ALL strings used by your module. These strings are represented by the $mod_strings variable accessed at any time after a global $mod_string call. The .html files located in this folder are used by the Help subsystem. Sugar provides the capabilities for multi-language support and dynamic loading via the admin panel of new language packs.
- metadata : Module-specific metadata files have been added to this folder with the addition of more metadata and extensibility into the Sugar platform. Most of the files and folders in this directory can be overridden by a file of the same name in the ./custom/<module>/metadata/ directory. Files and folders in this directory include:

    - additionaldetails.php : The content of the pop-up displayed in the list view
    - detailviewdefs.php : The field layout for the detail view
    - editviewdefs.php : The field layout for the edit view
    - listviewdefs.php : The base column layout displayed on the list view
    - popupdefs.php : The search fields and list columns for a module's pop-up view
    - quickcreatedefs.php : The layout for the quick create form utilized in multiple areas of the application
    - searchdefs.php : The field layout for the Basic Search and Advanced Search forms on the list view.
    - SearchFields.php : The mapping of definition of unique search fields in the module
    - studio.php : A mapping for Studio to identify files that override the standard metadata files for a module
    - subpaneldefs.php : The layout of the subpanels as they are shown on the detail view of the module
    - subpanels : A folder storing the definitions of the module's subpanels that will show on other module detail views when an appropriate relationship (one-to-many or many-to-many) exists
- tpls : The Smarty template files used for various the module header, footer, etc.
- views : This folder contains files that can override the default Model-View-Controller (MVC) framework view files. View files can perform multiple actions on the Smarty template or outputted HTML, allowing developers to modify and extend the default UI display classes and take full control of the user interface.

User Interface Framework

SugarCRM uses an implementation of the Model-View-Controller (MVC) pattern as the base of all application interactions. Working closely with the MVC framework is a metadata-driven UI framework where the high-level specification of parts of the user interface is described in a metadata structure.

Extension Framework

The extension framework in Sugar enables you to implement customizations of existing modules or create new modules. You can extend most of the functionality of Sugar through the various extension framework capabilities in an upgrade-safe manner. The Module Builder tool and Studio tool available in the Admin section enable you to make the customizations outlined below. You can then further extend your system by adding upgrade-safe custom code. The areas open to extension are:

- Modules : Create new modules and add them to Sugar
- Vardefs : Add custom fields to existing modules with the addition of your custom module
- Relationships : Create new relationships between your custom module(s) and existing modules
- Subpanels : Create/add new subpanel definitions to existing modules
- Strings : Override or add to module and application strings
- Menus : Override or add to Actions menus
- Layout Defs : Specify the displayed subpanels and the order in which they are displayed. Create the layout definition for a custom module and add it as a subpanel to the layout definition of an existing module.

Sugar Dashlets

Sugar Dashlets is a framework that provides for Sugar Dashlet containers to be included in the Sugar UI. Sugar Dashlet container objects display and interact with Sugar module data, with external sources such as RSS feeds, and with web services like Google Maps. Released originally in Sugar 4.5, Sugar Dashlets are a powerful new way to combine highly functional mash-ups in an attractive and easily tailored AJAX-based UI framework. Sugar Dashlets, located on the Sugar Home page, allow for the customization through simple drag-and-drop tools. The Sugar Dashlet Framework allows developers to easily create new Sugar Dashlets that can be installed in Sugar instances through the Module Loader.

Web Services

Sugar provides a Web Services API interface for developers to build integrations with Sugar for reading and writing data. Sugar provides Web Services APIs through the NuSOAP PHP implementation of the SOAP and REST protocol. SOAP (Simple Object Access Protocol) is used for making Remote Procedure Calls through the HTTP protocol by relaying messages in XML. The SugarSoap APIs, built on top of the NuSOAP PHP library, are included in the Sugar Community, Sugar Professional, Sugar Corporate, Sugar Enterprise, and Sugar Ultimate editions. REST (Representational State Transfer) is used for making method calls through HTTP protocol by sending and receiving messages in JSON/Serialize format. Framework supports the addition of multiple formats for REST. For example, you can add XML format to send and receive data.

Connectors

The Cloud Connector framework enables developers to integrate data from

external web services and widgets into their Sugar installation. Data from existing modules such as Accounts, Contacts, and Leads may act as inputs to retrieve external data.

The Connector classes inside of Sugar create a framework for integrating third party systems within Sugar. Currently there are three different base integrations. Documents allows Sugar to upload documents to a third party system, download documents from that system, and in future allow Sugar to setup sharing of those documents (if supported by the third party system). Meetings help Sugar integrate with external conference room systems or other meeting systems, enabling creation and editing of meetings, inviting attendees to the meeting, and including external meeting information in the email invitation. The Feed integration type allows Sugar to pull newsfeed data in from third party sources and display it in the My Activity Stream on the user's homepage.

For Community Edition, Sugar supports LinkedIn's Company Insider widget and InsideView. Use this as an example connector to learn the framework and create your own. Sugar Ultimate, Sugar Enterprise, Sugar Corporate, and Sugar Professional support additional connectors, such as Hoovers , Zoominfo, Google Docs, Twitter, Facebook, GoToMeeting, IBM SmartCloud, and WebEx, and have the ability to merge the data into existing Sugar records.

The main components for the framework are factories, source, and formatter classes.

Factories return appropriate source or formatter instance for a connector. Sources encapsulate the retrieval of data as a single record, or a list, or records of the connectors. Formatters render the display elements of the connectors. For more information, see the expanded section on Connectors .

Last Modified: 01/15/2016 10:01pm

Application Framework

Application Framework.

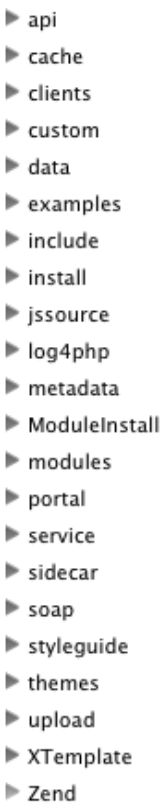
Last Modified: 09/26/2015 04:20pm

Introduction

Application Framework

The Sugar application code is based on a modular framework with secure entry points into the application (e.g. index.php or soap.php). All modules, core or custom, must exist in the <sugar root>/modules/ folder. Modules represent business entities or objects in Sugar such as Contacts, and the object has fields or attributes that are stored in the database, as well as a user interface (UI) for the user to create and modify records. A module encompasses definitions for the data schema, user interface, and application functionality.

The structure of Sugar's root directory is shown below:



Directory Structure

SugarCRM code resides in various directories within the Sugar installation. The structure of the directories within the Sugar application consists of the following root level directories:

- api : v6+ REST API.
- cache : Various cache files written to the file system to minimize database accesses and store user interface templates created from metadata.
- clients : Files needed for the new view frameworks in 6.6+
- custom : Location for upgrade-safe customizations such as custom field definitions, user interface layouts, and business logic hooks

- data : Key system files, most notably the SugarBean base class which controls the default application logic for all business objects in Sugar
- examples : Examples of how to extend Sugar, never run from the main app and should be removed from production instances
- include : Many Sugar utility functions as well as other libraries that Sugar utilizes as part of its operations are located here. Most notable in this directory is the utils.php file that contains the most widely used utility functions.
- jssource : Non-minified versions of JS files
- log4php : This has been deprecated as it has been replaced by LoggerManager.
- metadata : Relationship metadata for all many-to-many data relationships between the business objects
- ModuleInstall : Code used to support the Sugar Module Loader
- modules : Contains all modules in the system. Custom modules installed through the Module Loader exist here as well.
- portal : Customer portal code v1
- portal2 : Customer portal code v2 for 6.6+ (Enterprise and Ultimate only)
- service : Web Services code for SOAP and REST v2 - v4_1
- sidecar : Core files used for the sidecar framework
- soap : Web Services code for SOAP v1; this has been deprecated.
- styleguide : Style guide that the entire application adheres to for a consistent presentation
- themes : Files used to display the various themes
- upload : Files uploaded into the application such as Note Attachments or Documents reside in this directory (refer to upload_dir parameter in the config.php file)
- XTemplate : Old templating library that is being phased out and replaced by Smarty, but some code still may use it.
- Zend : Classes used in the framework

Key Concepts

These are the main files, classes and application concepts that comprise the Sugar platform.

Application Concepts

- Controller : Directs all incoming page requests. This can be overridden in each module to change the default behavior and relies on Entry point parameters (described below) to serve the appropriate page.
- Views : A set of user interface actions managed by the Controller, the default views in Sugar include the Detail View, Edit View, and List View.

- Display strings : Sugar is fully internationalized and localizable. Every language pack has its own set of display strings which is the basis of language localization. There are two types of display strings in the Sugar application: application strings and module strings.
    - Application strings contain the user interface labels displayed globally throughout the application.
        - $GLOBALS['app_strings']
            - Array - Contains labels values
        - $GLOBALS['app_list_strings']
            - Array - Contains the system-wide drop-down list values.
    - Each language has its own application strings variables.
        - $GLOBALS['mod_strings']
            - Array - Contains strings specific to the current, or in-focus, module.
- Dropdown lists : Dropdown lists are represented as name => value array pairs located in the application strings mentioned above. The name value is stored in the database where the value is displayed in the Sugar User Interface (UI). Sugar enables you to create and edit drop-down lists and their values through the UI in Studio. Use the handy get_select_options_with_id() utility function to help render the <select> input options to work with drop-down lists in Edit Views. Use the translate() utility function to translate the string key you are working with into the user's currently selected display language.

Files

- SugarBean.php : This file located under the <sugar root>/data folder contains the SugarBean base class used by all business entity or module objects. Any module that reads, writes, or displays data will extend this class. The SugarBean performs all of the heavy lifting for data interactions, relationship handling, etc.
- modules.php : The modules.php file contains several variables that define which modules are active and usable in the application.

Variables

- $dictionary : The $dictionary array contains all module field variable definitions (vardefs), as well as the relationship metadata for all tables in the database. This array is dynamically built based upon the vardefs.php definitions.

Entry Points

The primary user interface entry point for Sugar is through index.php located in the root Sugar folder. The main parameters for most calls are:

- module : The module to be accessed as part of the call
- action : The action within the module
- record : The record ID

The following is a sample URL for a Sugar call:

```
http://{sugar_url}/index.php?module=Contacts&action=DetailView&record=
d545d1dd-0cb2-d614-3430-45df72473cfb
```

This URL invokes the Detail View action from within the Contacts module to display the record denoted by the record request value.

Other commonly used parameters are return_module , return_action, and return_id. This group of request parameters is used when a user cancels an action such as creating or editing a record.

Note: As of Sugar 5.1, entry points were consolidated into index.php. Previous versions had other files as entry points into the application.

Last Modified: 01/15/2016 09:52pm

ACL

Overview

ACLs and restricting access.

Access Control Lists

ACLs or Access Control Lists, are used to restrict access to modules, data and actions available to users within Sugar. ACLs are defined in the Roles module which can be found in the Admin section. ACLs are not available in the Community Edition of Sugar.

Checking Access

You can verify role access to content by using the checkAccess() method found in the ACLController.

Parameters

- $category : Corresponds to the module directory where the bean resides. For example: Accounts.
- $action : The action you want to check against. For example, Edit. These actions correspond to actions in the acl_actions table as well as actions performed by the user within the application.
- $is_owner : verifies if the owner of the record is attempting an action. Defaults to false. This is relevant when the access level = ALLOW_OWNER.
- $type : Defaults to "module".

Example

```
if (ACLController::checkAccess($category, $action, $is_owner, $type))

{

    //Code

}
```

See the Role Management documentation for a list of actions and their possible values.

Last Modified: 10/11/2016 06:44pm

Administration Links

Overview

Managing administration links.

Administration Links

Administration links are the links located in the admin section of Sugar. As of 6.3.x, administration links can be created using the extension framework. The

global links extension directory is located at ./ custom/Extension/modules/Administration/Ext/Administration/. PHP files found is this directory will be compiled into ./custom/modules/Administration/Ext/Administration/administration.ext.php after a Quick Repair and Rebuild.Additional information on this can be found in the extensions Administration section. The current links defined in the administration section can be found in . /modules/Administration/metadata/adminpaneldefs.php.

Example

The following example will create a new admin panel:

./custom/Extension/modules/Administration/Ext/Administration/<file>.php

```
<?php

    $admin_option_defs = array();
    $admin_option_defs['Administration']['<section key>'] = array(
        //Icon name. Available icons are located in
./themes/default/images
        'Administration',

        //Link name label
        'LBL_LINK_NAME',

        //Link description label
        'LBL_LINK_DESCRIPTION',

        //Link URL
        './index.php?module=<module>&action=<action>',
    );

    $admin_group_header[] = array(
        //Section header label
        'LBL_SECTION_HEADER',

        //$other_text parameter for get_form_header()
        '',

        //$show_help parameter for get_form_header()
        false,

        //Section links
        $admin_option_defs,
```

```
        //Section description label
        'LBL_SECTION_DESCRIPTION'
    );
```

Next, we will populate the panel label values:

./custom/Extension/modules/Administration/Ext/Language/en_us.<name>.php

```
<?php

    $mod_strings['LBL_LINK_NAME'] = 'Link Name';
    $mod_strings['LBL_LINK_DESCRIPTION'] = 'Link Description';
    $mod_strings['LBL_SECTION_HEADER'] = 'Section Header';
    $mod_strings['LBL_SECTION_DESCRIPTION'] = 'Section Description';
```

Finally, navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions and the panel will appear in the Admin section.

Last Modified: 09/26/2015 04:20pm

AjaxUI

Overview

Overview of the AjaxUI features and design considerations.

About the AjaxUI

In version 6.3 we are introduced a new method for loading pages that uses AJAX to load the html content rather than doing a full page refresh. This has some major benefits regarding performance such as:

Reduced server load

The server no longer has to render the header, footer, or check the standard javascript, image, and style files on each request. Only the initial load will render these elements.

No Full Rebuild of the Dom

Rather than destroying and rebuilding the entire Dom tree, only the center content area is updated.

Script Files are not Parsed

Normally when a page is loaded, even if the javascript files are cached, the browser must reload into memory and parse all of the script files. Sugar uses a very large amount of base javascript that remains the same on each page. By only parsing the javascript specific to the current page, a huge performance boost is gained. This is especially noticeable on machines with fast net connections but slow drive/cpu speeds.

Fewer Requests Per Page

When a web server is configured to for very large cache timeouts, this isn't a very big deal. The client will only load the script, image, and style files once and never make a request for them again until Sugar is updated. However, many servers are not configured this way and the client will make a request to the server, even if it is just to verify the cache isn't out of date for every single resource on a page. This can mean up to 100 requests for some pages. With the AjaxUI, these files stay in memory until the user closes the browser even if the cache expire is not configured. For a connection with a high ping rate, it can decrease page load time from tens of seconds for 50 requests down to a single request taking under a second.

Side Effects

There are some side effects of the AjaxUI that can cause issues with customizations.

document.write

Because the Dom has already "fully loaded" when the page is rendered, any custom javascript that uses document.write will replace the current content instead of appending to the current page. The best practice here is to use Dom Manipulation or at worst, append to document.body.innerHTML - but this is also not recommended.

onLoad

Similar to document.write, this event is no longer reliable because the page has already "loaded" before the main content is inserted. If your javascript relies on some content in the page to finish rendering before it can fire, you should wrap your code with YAHOO.util.Event.onContentReady.

Page Order vs Execution Order

One final javascript issue is the order of execution of script blocks. This isn't always a problem but it tends to be browser specific. If for instance you wanted to use jquery and your code looked something like this:

```
<script type="text/javascript" src="include/javascript/jquery.js" />
<script type="text/javascript">
    $("#myDiv").append("Hello World");
</script>
```

You cannot guarantee that the second bock will fire after jquery has loaded. To help with this problem, we've added a function to Sugar called SUGAR.util.doWhen. It has an signature similar to onContentReady except that instead of only wait for Dom elements to appear, it can wait for any condition to be true. The first parameter can either be a string to be evaluated or a function to call, but as soon as either is true, the function passed as the second parameter is fired. So using doWhen, the above block could either look like:

```
<script type="text/javascript" src="include/javascript/jquery.js" />
<script type="text/javascript">
    SUGAR.util.doWhen("typeof $ != 'undefined'", function(){
        $("#myDiv").append("Hello World");
    });
</script>
```

or

```
<script type="text/javascript" src="include/javascript/jquery.js" />
<script type="text/javascript">
    SUGAR.util.doWhen(function(){ return (typeof $ != 'undefined')},
function(){
        $("#myDiv").append("Hello World");
    });
</script>
```

or even

```
<script type="text/javascript" src="include/javascript/jquery.js" />
```

```
<script type="text/javascript">
    var checkJQLoaded = function(){ return (typeof $ != 'undefined')};
    var myScope = {target:"#myDiv"};
    SUGAR.util.doWhen(checkJQLoaded, function(toAppend){
        $(this.target).append(toAppend);
    }, "Hello World", myScope);
</script>
```

JSON Responses

The final thing to watch out for is php warnings and errors causing invalid server responses. These kinds of errors being displayed won't cause the same level of headache with a normal UI as the user can often just ignore them and continue until the admin gets around to fixing them. With the AjaxUI, the errors can cause the server response to become invalid and the page fail to load entirely. On a production instance these errors and warnings should be logged rather than shown to the user anyhow. Also, if you have a customization that tries to bypass the MVC Controller and send content directly, it could also cause display issues.

Last Modified: 09/26/2015 04:20pm

Authentication

Overview of Authentication.

Last Modified: 11/18/2015 01:08am

Oauth

Overview

OAuth(Open Authorization) is an open standard for authorization users across software applications. SugarCRM implements Oauth in order to integrate with other applications such as IBM Smart Cloud.

Using OAuth with Sugar

Step 1: Establishing Consumer Key

You need to create a Consumer Key/Secret Pair in 'Admin > OAuth Keys' page to use the Sugar OAuth provider. The key pair can be arbitrary strings and should be used by your client when calling OAuth functions.

Step 2: Creating Request Token

Make a REST call to oauth_request_token method (supported in REST API version 4+) to create a Request Token.

Below is an example using PHP OAuth extension:

```
$oauth = new OAuth($customKey, $customSecret,
OAUTH_SIG_METHOD_HMACSHA1, OAUTH_AUTH_TYPE_URI);
$url = 'http://{sugar_url}/service/v4_1/rest.php';
$request_token_info =
$oauth->getRequestToken($url."?method=oauth_request_token");
```

The response for this method is the query-encoded string containing the following three parameters:

- oauth_token
- oauth_token_secret
- authorize_url

Example:

```
oauth_token=bf1492236fbe&oauth_token_secret=5b05d09a0b7e&oauth_callbac
k_confirmed=true&authorize_url=http%3A%2F%2Fmysugarinstance.com%2Finde
x.php%3Fmodule%3DOAuthTokens%26action%3Dauthorize
```

The PHP OAuth extension method getRequestToken automatically parses the string and returns it as an array, other clients parse the string manually.

Step 3: Approve Request Token

The Request Token should be approved manually by the user. To achieve this, the user should log into Sugar and then go to the URL produced by adding token to the authorize URL returned above, e.g.:

```
http://{sugar_url}/index.php?module=OAuthTokens&action=authorize&token
```

```
=bf1492236fbe
```

The client should produce this URL for the user with the information returned in the previous step. The user then receives the verification code required to input in the client application.

Step 4: Request Access Token

The client should use the token and secret received in Step 2 and the verifier that the user received in Step 3 to request the Access Token, using the oauth_access_token method. Example using PHP OAuth extension:

```
$oauth = new OAuth($customKey, $customSecret,
OAUTH_SIG_METHOD_HMACSHA1, OAUTH_AUTH_TYPE_URI);
$url = 'http://{sugar_url}/service/v4_1/rest.php';
$oauth->setToken($token, $secret);
$access_token_info =
$oauth->getAccessToken($url."?method=oauth_access_token&oauth_verifier
=$verify");
```

The response for this contains the OAuth Access Token and secret, query-encoded.

Example:

```
oauth_token=bf1492236fbe&oauth_token_secret=5b05d09a0b7e
```

Again, the PHP OAuth extension method getAccessToken automatically parses the string and returns it as an array; other clients parse the string manually.

Step 5: Using Access Token

Access token can be used either directly to access REST API methods via OAuth, or to establish a login session.

Use the access token directly to access any method via OAuth:

```
$oauth = new OAuth($customKey, $customSecret,
OAUTH_SIG_METHOD_HMACSHA1, OAUTH_AUTH_TYPE_URI);
$url = 'http://{sugar_url}/service/v4_1/rest.php';
$oauth->setToken($token, $secret);
$data =
$oauth->fetch($url."?method=get_available_modules&input_type=JSON&requ
est_type=JSON&response_type=JSON");
```

Use the recommended oauth_access method to establish a new session (similar to username/password login):

```
$oauth = new OAuth($customKey, $customSecret,
OAUTH_SIG_METHOD_HMACSHA1, OAUTH_AUTH_TYPE_URI);
$url = 'http://{sugar_url}/service/v4_1/rest.php';
$oauth->setToken($token, $secret);
$data =
$oauth->fetch($url."?method=oauth_access&input_type=JSON&request_type=
JSON&response_type=JSON");
```

You will receive a JSON response which will have session ID as id value. Use this ID as session parameter to other calls.

Last Modified: 06/30/2016 05:41pm

Charts

Overview of charts.

Last Modified: 11/18/2015 01:08am

Custom Charts

Overview

Provides technical overview and know-how of a technology related to the product or a particular feature.

Custom Charts

To create custom charts in SugarCRM you need to create a SugarChart subclass that will be returned from the SugarChartFactory. When charts are rendered in the system, a call is made to the static getInstance method of SugarChartFactory. The getInstance method accepts two parameters:

- a string name of the chart engine
- an optional string name of a module

The chart engine value is retrieved from the Sugar configuration settings. The optional module name supports modules that require additional data processing and rendering. Sugar comes with a JitReport class specifically to handle the nuances of rendering charts for the Reports module.
In the following example, we highlight how to change the formatting of the chart to move the legend to the top of the chart (by default, it is rendered on the bottom of the chart).
The first step is to create two folders:

- ./custom/include/SugarCharts/CustomJit
- ./custom/include/SugarCharts/tpls

In the ./custom/include/SugarCharts/CustomJit folder, create two PHP files:

- CustomJit.php
- CustomJitReports.php

In our trivial example, we will simply override the display method for both classes to point to our new custom template so that we may move the legend to the top of the chart.
See below for the code in the CustomJit.php file:

```
<?php
    require_once("include/SugarCharts/Jit/Jit.php");
    class CustomJit extends Jit
    {
        function display($name, $xmlFile, $width='320', $height='480',
$resize=false)
        {
            parent::display($name, $xmlFile, $width, $height,
$resize);
            return
$this->ss->fetch('custom/include/SugarCharts/CustomJit/tpls/chart.tpl'
);
        }
    }

?>
```

Copy over the file from ./include/SugarCharts/Jit/tpls/chart.tpl to the ./custom/include/SugarCharts/CustomJit/tpls directory. We will change the Smarty code that outputs the legend to be moved to the top. Here's the snipped-off code changed from chart.tpl:

```
<div class="chartContainer">
    <div id="sb{$chartId}" class="scrollBars">
        <div id="legend{$chartId}" class="legend"></div>
        <div id="{$chartId}" class="chartCanvas" style="width:
```

```
{$width}; height: {$height}px;"></div>
    </div>
</div>
```

Finally, all that's left is to change your configuration file settings so that the chartEngine value is CustomJit. The next time you render a chart with a legend, the legend will appear at the top of the chart.

Last Modified: 09/26/2015 04:20pm

Connectors

Overview of connectors.

Last Modified: 11/18/2015 01:08am

Introduction

This section covers the design specifications for the connector integration capabilities in Sugar called Connectors. The Connector framework allows for various data retrieved through REST and SOAP protocols to be easily viewed and entered into SugarCRM.

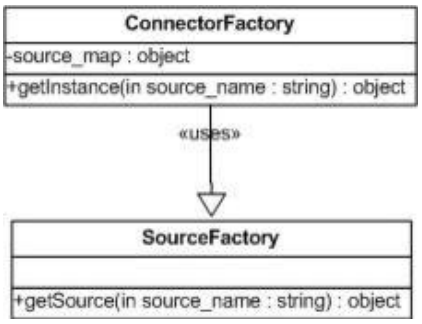
The Connector framework is designed to provide an abstract layer around a connector. So, essentially our own database would just be considered another connector along with any SOAP or REST connector. These connectors, in theory, can then be swapped in and out seamlessly. Hence, providing the framework for it and leveraging a small component is a step in the right direction.The connectors can then be loaded in by a factory, returned, and called based on their interface methods.

The main components for the connector framework are the factories, source, and formatter classes. The factories are responsible for returning the appropriate source or formatter instance for a connector. The sources are responsible for encapsulating the retrieval of the data as a single record or a list or records of the connectors. The formatters are responsible for rendering the display elements of the connectors.

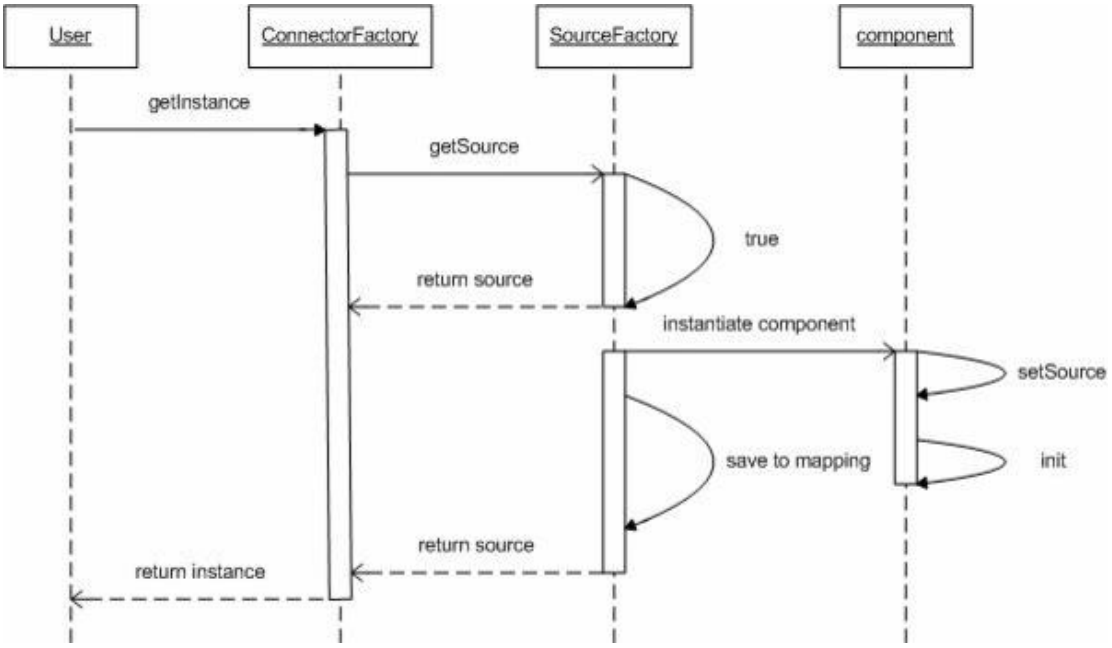
Last Modified: 11/18/2015 01:08am

Factories

The primary factory is the ConnectorFactory class. It uses the static SourceFactory class to return a connector instance.



The main points to note are that given a source name (e.g. "ext_soap_hoovers"), the underscore characters are replaced with the file separator character. In this case, "ext_soap_hoovers" becomes "ext/soap/hoovers". Then the SourceFactory first scans the ./modules/Connectors/connectors/sources and then the ./custom/modules/Connectors/connectors/sources directories to check for the presence of this file, and returns a source instance if the file is found. The following sequence diagram attempts to highlight the aforementioned steps.



Last Modified: 09/26/2015 04:20pm

Sources

The sources are the centerpiece of the Connectors framework. There are two

categories of sources - REST implementations and SOAP implementations. The default source class is abstract and subclasses need to override the getList and getItem methods. The class name of the source should be prefixed with either "ext_soap_" or "ext_rest_". This is because the "_" character serves as a delimiter into the file system for the class to be found. For example, a SOAP implementation for a source we call "Test" will have the class name "ext_soap_test" and a REST implementation will have the class name "ext_rest_test".

```
/** getItem
* Returns an array containing a key/value pair(s) of a source record
* @param Array $args Array of arguments to search/filter by
* @param String $module String optional value of the module that the
connector is attempting to map to
* @return Array of key/value pair(s) of the source record; empty Array
if no results are found
*/
public function getItem($args=array(), $module=null){}

/** getList
* Returns a nested array containing a key/value pair(s) of a source
record
* @param Array $args Array of arguments to search/filter by
* @param String $module String optional value of the module that the
connector is attempting to map to
* @return Array of key/value pair(s) of source record; empty Array if
no results are found
*/
public function getList($args=array(), $module=null){}
```

Here is an example of the Array format for the getItem method of the Test source:

```
Array(
  ['id'] => 19303193202,
  ['duns'] => 19303193202,
  ['recname'] => 'SugarCRM, Inc',
  ['addrcity'] => 'Cupertino',
)
```

Here is an example of the Array format for the getList method of the Test source:

```
Array(
  [19303193202] => Array(
    ['id'] => 19303193202,
    ['duns'] => 19303193202,
    ['recname'] => 'SugarCRM, Inc',
    ['addrcity'] => 'Cupertino',
```

```
  ),
  [39203032990] => Array(
    ['id'] => 39203032990,
    ['duns'] => 39203032990,
    ['recname'] => 'Google',
    ['addrcity'] => 'Mountain View',
  )
)
```

The key values for the getList/getItem entries should be mapped to a vardefs.php file contained with the source. This vardefs.php file is required. In this case, we have something like:

```
<?php

$dictionary['ext_rest_test'] = array(
    'comment' => 'vardefs for test connector',
    'fields' => array (
        'id' => array (
            'name' => 'id',
            'vname' => 'LBL_ID',
            'type' => 'id',
            'hidden' => true
            'comment' => 'Unique identifier'
        ),
        'addrcity' => array (
            'name' => 'addrcity',
            'input' => 'bal.location.city',
            'vname' => 'LBL_CITY',
            'type' => 'varchar',
            'comment' => 'The city address for the company',
            'options' => 'addrcity_dom',
            'search' => true,
        ),
    )
);

?>
```

Note the 'input' key for the addrcity entry. The 'input' key allows for some internal argument mapping conversion that the source uses. The period (.) is used to delimit the input value into an Array. In the example of the addrcity entry, the value bal.location.city will be translated into the Array argument ['bal']['location']['city'].

The 'search' key for the addrcity entry may be used for the search form in the

Connectors' data merge wizard screens available for the Ultimate, Enterprise, Corporate, and Professional editions.

Finally, note the 'options' key for the addrcity entry. This 'options' key maps to an entry in the mapping.php file to assist in the conversion of source values to the database value format in SugarCRM. For example, assume a source that returns American city values as a numerical value (San Jose = 001, San Francisco = 032, etc.). Internally, the Sugar system may use the city airport codes (San Jose = sjc, San Francisco = sfo). To allow the connector framework to map the values, the options configuration is used.

Sources also need to provide a config.php file that may contain optional runtime properties such as the URL of the SOAP WSDL file, API keys, etc. These runtime properties shall be placed under the 'properties' Array. At a minimum, a 'name' key should be provided for the source.

```
<?php

    $config = array (
        'name' => 'Test', //Name of the source
        'properties' =>  array (
            'TEST_ENDPOINT' =>
'http://test-dev.com/axis2/services/AccessTest',
            'TEST_WSDL' => 'http://hapi-dev.test.com/axis2/test.wsdl',
            'TEST_API_KEY' => 'abc123',
        ),
    );
?>
```

An optional mapping.php file may be provided so that default mappings are defined. These mappings assist the connector framework's component class. In the component class there are the fillBean and fillBeans methods. These methods use the getItem/getList results from the source instance respectively and return a SugarBean/SugarBeans that map the resulting values from the source to fields of the SugarBean(s). While the mapping.php file is optional, it should be created if the 'options' key is used in vardefs entries in the vardefs.php file.

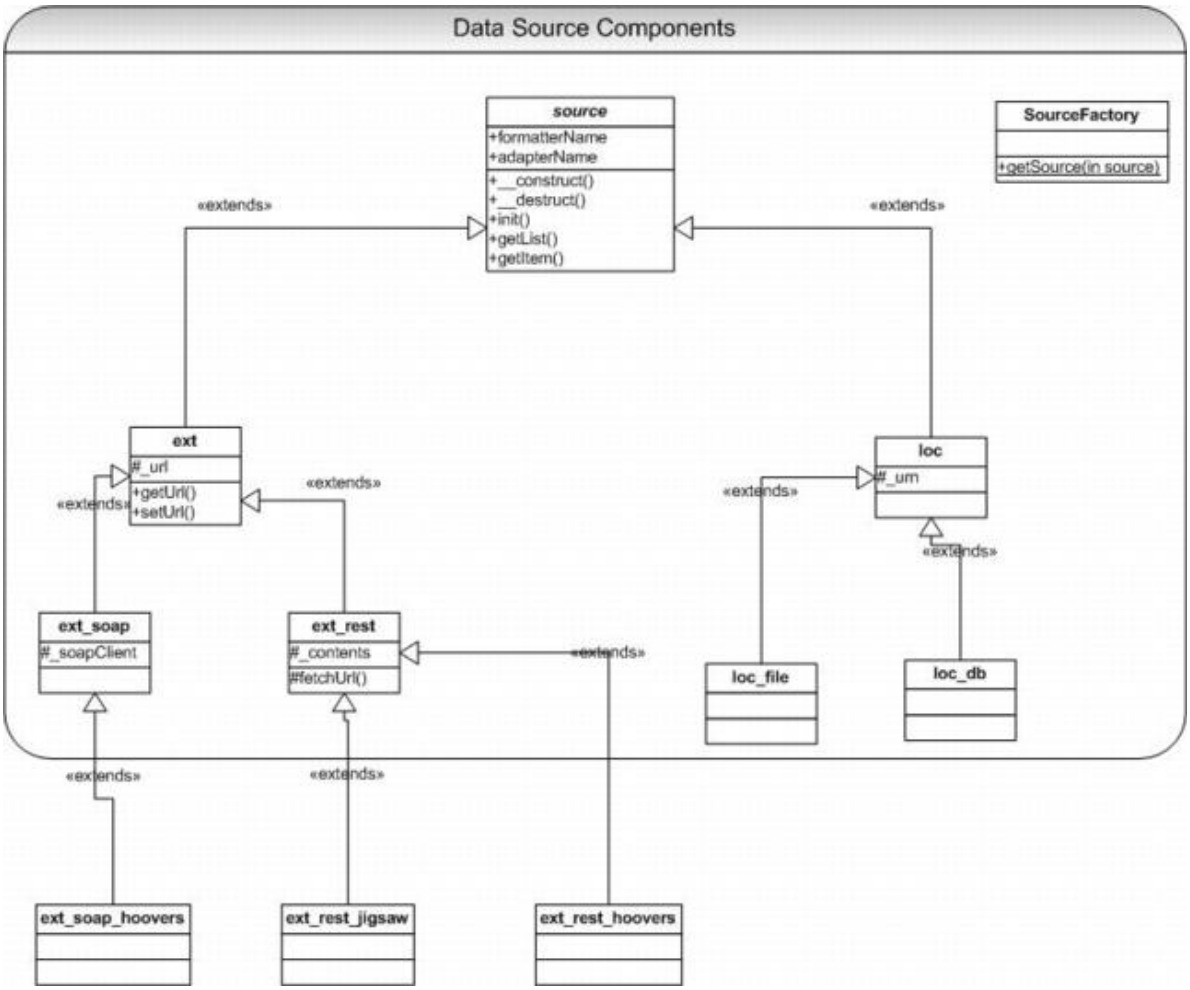
```
<?php

$mapping = array(
    'beans' => array(
        'Leads' => array(
            'id' => 'id',
            'addrcity' => 'primary_address_city',
        ),
        'Accounts' => array(
```

```
            'id' => 'id',
            'addrcity' => 'primary_address_city',
        ),
    ),
    'options' => array(
        'addrcity_dom' => array(
            '001' => 'sjc', //San Jose
            '032' => 'sfo', //San Francisco
        ),
    ),
);

?>
```

In this example, there are two modules that are mapped (Leads and Accounts). The source keys are the array keys while the Sugar module's fields are the values. Also note the example of the 'options' array as discussed in the vardefs.php file section. Here we have defined an addrcity_dom element to map numerical city values to airport codes.

The source file (test.php) and its supporting files will be placed into self contained directories. In our test example, the contents would be as follows:

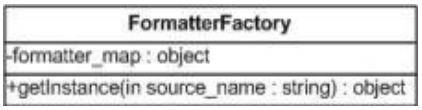
- ./custom/modules/Connectors/connectors/sources/ext/rest/test/test.php
- ./custom/modules/Connectors/connectors/sources/ext/rest/test/vardefs.php
- ./custom/modules/Connectors/connectors/sources/ext/rest/test/config.php
- ./custom/modules/Connectors/connectors/sources/ext/rest/test/mapping.php



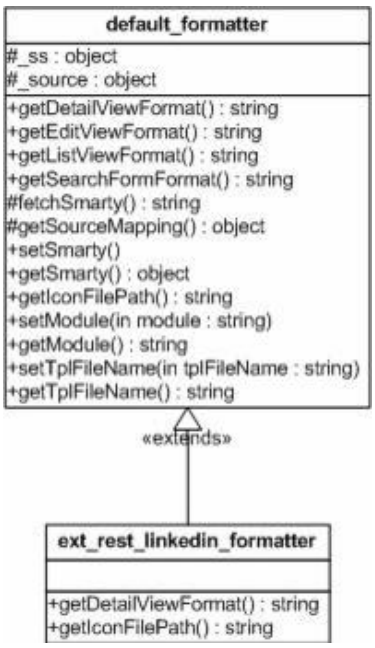
Last Modified: 09/26/2015 04:20pm

Formatters

The optional formatter components are used by the connector framework to render a widget that may display additional details and information. Currently, they are shown in the detail view screens for modules that are enabled for the connector. Similar to the source class, the formatter class has a corresponding factory class (FormatterFactory). The formatters also follow the same convention of using the "ext_rest_" or "ext_soap_" prefix. However, to distinguish conflicting class names, a suffix "_formatter" is also used. Formatters extend from default_formatter. Retrieving a formatter instance is similar to retrieving a source instance except that the FormatterFactory scans in order the ./modules/Connectors/connectors/formatters first and then the ./custom/modules/Connectors/connectors/formatters directories.



The following class diagram shows an example of the LinkedIn formatter extending from the default formatter.



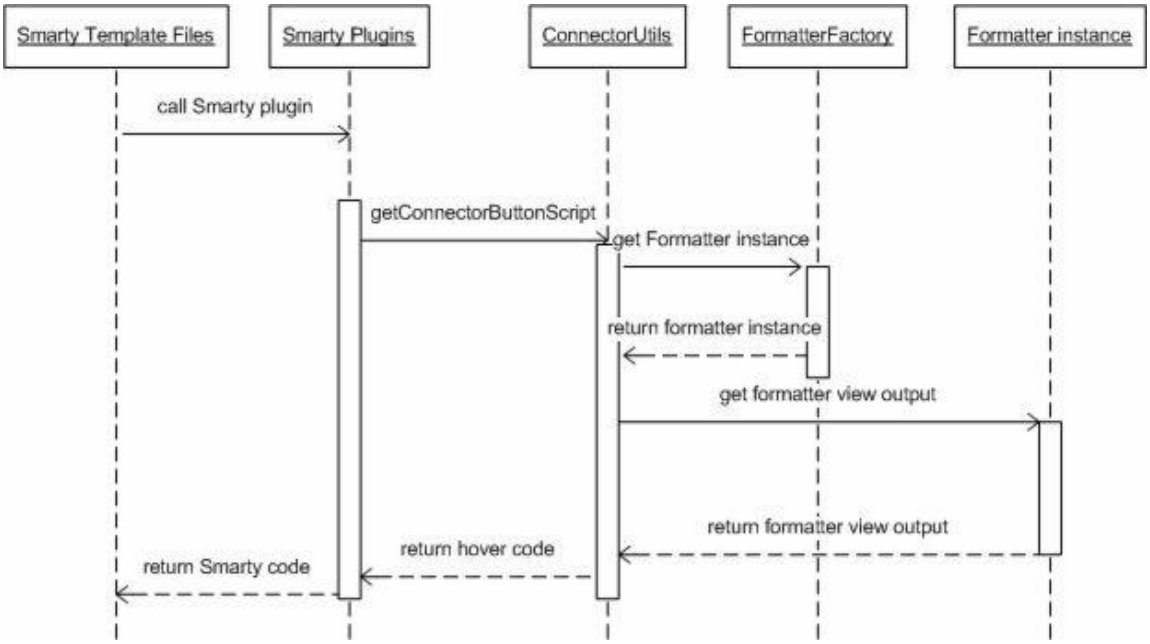
Here, we have a subclass ext_rest_linkedin_formatter that overrides the getDetailViewFormat and getIconFilePath methods.

```
<?php

if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

require_once('include/connectors/formatters/default/formatter.php');

class ext_rest_linkedin_formatter extends default_formatter {

    public function getDetailViewFormat() {
        $mapping = $this->getSourceMapping();
        $mapping_name =
!empty($mapping['beans'][$this->_module]['name']) ?
$mapping['beans'][$this->_module]['name'] : '';

        if(!empty($mapping_name)) {
            $this->_ss->assign('mapping_name', $mapping_name);
            return $this->fetchSmarty();
        }


$GLOBALS['log']->error($GLOBALS['app_strings']['ERR_MISSING_MAPPING_EN
TRY_FORM_MODULE']);
```

```
        return '';
    }

    public function getIconFilePath() {
        return
'modules/Connectors/connectors/formatters/ext/rest/linkedin/tpls/linke
din.gif';
    }
}
```

The default_formatter class provides an implementation of the getDetailViewFormat method. This method is responsible for rendering the hover code that appears next to certain Detail View fields. The default_formatter class will scan the tpls directory for a Smarty template file named after the module that is being viewed. For example, the file *formatters/ext/rest/linkedin/tpls/Accounts.tpl will be used for the Accounts popup view if the file exists. If the module named template file is not found, it will attempt to use a file named default.tpl.

Formatters are invoked from the Smarty template code, which in turn uses Smarty plugins to call the Formatter classes. The following sequence diagram illustrates this.



Last Modified: 09/26/2015 04:20pm

Class Definitions

Classes should be named "ExtAPI" and should be placed in the

./include/externalAPI/ directory (the directory name should be ./custom/include/externalAPI/ if this is externally created code) with the filename of "ExtAPI.php". There can be other files located in the directory with your class and they will be ignored by the application, the external api class may use these for additional libraries, XML/XSL templates, or any other necessory file. In order for the new external API class to be picked up by the system, it is necessary to flush the external API cache of class names by going to the "Admin" module, selecting the "Repair" link and then selecting "Quick Repair and Rebuild", after this is complete any new external API classes that match the naming convention.

Classes based on OAuth methods for authentication and access control should extend the "OAuthPluginBase" class located at ./include/externalAPI/Base/OAuthPluginBase.php. Username and password based API's should extend the "ExternalAPIBase" class located at ./include/externalAPI/Base/ExternalAPIBase.php. Classes should implement the ExternalAPIPlugin API, along with the WebDocument, WebFeed and/or WebMeeting API's depending on their feature set. Classes may implement more than one of the WebDocument/WebFeed/WebMeeting API's simultaneously and there should be only one external API class per 3rd party site so a user does not have to authenticate against the same site more than once.

Method calls almost always return an array containing at least a "success" element, if this element is true it can be assumed that the method call was successful. If the method returns a "success" value of false, an additional element of "errorMessage" must be set with a user-readable error message explaining the problem.

Classes may optionally use the Sugar "Connectors" to allow for site-wide configuration of the class, things such as API keys and system wide URL's may be entered there. Creating a sub-connector and linking it to the class is beyond the scope of this document, for a reference implementation you will want to check the Facebook external API.

Last Modified: 11/17/2015 11:37pm

Examples

Examples of working with connectors

Last Modified: 11/18/2015 01:08am

Creating A Custom Connector

Overview

Describes all the various files needed when creating a connector. A downloadable example is attached to this article.

Connectors

The minimal files a connector should provide are a config file (config.php), a variable definition file (vardefs.php), a language file (<language>.lang.php) and a connector source file (<connector name>.php). An optional default mapping file that associates your connector's fields with fields in Sugar's modules may be provided (mapping.php). You will also need to choose a connector protocol type of REST or SOAP. This will be reflected in the directory the connector resides in of 'rest' or 'soap'.

Your directory should look like this:

- ./custom/modules/Connectors/connectors/sources/ext/<protocol type>/example/source/<connector name>.php
- ./custom/modules/Connectors/connectors/sources/ext/<protocol type>/example/source/vardefs.php
- ../custom/modules/Connectors/connectors/sources/ext/<protocol type>/example/config.php
- ./custom/modules/Connectors/connectors/sources/ext/<protocol type>/example/language/<lang>.lang.php
    - (default language file for localization)
- ./custom/modules/Connectors/connectors/sources/ext/<protocol type>/example/source/mapping.php
    - (optional)

Protocol Type

The first step is to determine the connector protocol type to create the connector source file. Currently the two Web Services protocols supported are REST and SOAP. If the web service is a REST protocol, then the connector should extend the ext_rest class defined in the file ./include/connectors/sources/ext/rest/rest.php. The class name should contain the class name ext_rest_<suffix>.

```
<?php
     require_once('include/connectors/sources/ext/rest/rest.php');
     class ext_rest_example extends ext_rest
```

```
     {

     }
?>
```

If the web service is a SOAP protocol, then the connector should extend the ext_soap class defined in the file ./include/connectors/sources/ext/soap/soap.php. The class name should contain the class name ext_soap_<suffix>

```
<?php
     require_once('include/connectors/sources/ext/soap/soap.php');
     class ext_soap_example extends ext_soap
     {

     }
?>
```

Source

The next step in creating a connector is to create the source. This example connector will be created as a REST connector in the directory ./custom/modules/Connectors/connectors/sources/ext/rest/example/ that extends ./include/connectors/sources/ext/rest/rest.php.

Connector Class (.php)

This connector class will be named 'ext_rest_example'. This class will reside in the file 'example.php' and will provide implementations for getItem() and getList() methods. These are the two methods that a connector must override and provide an implementation for. They are called by the component class (include/connectors/component.php). The getList() method as its name suggests, returns a list of results for a given set of search criteria that your connector can handle. The getItem() method will return a single connector record. For example, if your connector is a person lookup service, the getList() method may return matching person values based on a first and last name search. The getItem() method should return values for a unique person. Your service may uniquely identify a person based on an internal id or perhaps an email address.

getList()

The getList() method accepts two arguments. $args is an Array of argument values and $module is a String value of the module that the connector framework is

interacting with. The getList() method should return a multi-dimensional Array with each record's unique id as the key. The value should be another Array of key/value pairs where the keys are the field names as defined in the vardefs.php file.

```
public function getList($args=array(), $module=null)
{
    $results = array();
    if(!empty($args['name']['last']) &&
strtolower($args['name']['last']) == 'doe')
    {
        $results[1] = array('
            id' => 1,
            'firstname' => 'John',
            'lastname' => 'Doe',
            'website' => 'www.johndoe.com',
            'state' => 'CA'
        );

        $results[2] = array(
            'id' => 2,
            'firstname' => 'Jane',
            'lastname' => 'Doe',
            'website' => 'www.janedoe.com',
            'state' => 'HI'
        );
    }

    return $results;
}
```

getItem()

The getItem() method also accepts two arguments. $args is an Array of argument values and $module is a String value of the module that the connector framework is interacting with. The getItem() method will be called with a unique id as defined in the getList() method's results.

```
public function getItem($args=array(), $module=null)
{
    $result = null;
    if ($args['id'] == 1)
    {
        $result = array();
        $result['id'] = '1'; //Unique record identifier
```

```
        $result['firstname'] = 'John';
        $result['lastname'] = 'Doe';
        $result['website'] = 'http://www.johndoe.com';
        $result['state'] = 'CA';
    }
    else if ($args['id'] == 2)
    {
        $result = array();
        $result['id'] = '2'; //Unique record identifier
        $result['firstname'] = 'Jane';
        $result['lastname'] = 'Doe';
        $result['website'] = 'http://www.janedoe.com';
        $result['state'] = 'HI';
    }

    return $result;
}
```

test()

This is an optional step where you may wish to provide functionality for your connector so that it may be tested through the administration interface under the "Set Connector Properties" section. To enable testing for your connector, set the connector class variable _has_testing_enabled to true in the constructor and provide a test () method implementation.

```
public function __construct()
{
    parent::__construct();
    $this->_has_testing_enabled = true;
}

public function test()
{
    $item = $this->getItem(array('id'=>'1'));
    return !empty($item['firstname']) && ($item['firstname'] ==
'John');
}
```

You should also note that to enable the formatter hover component, you will need to also add the following to the construct:

```
$this->_enable_in_hover = true;
```

Class Example

The full 'ext_rest_example' class is shown below:

./custom/modules/Connectors/connectors/sources/ext/rest/example/example.php

```
<?php

    if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    require_once('include/connectors/sources/ext/rest/rest.php');

    class ext_rest_example extends ext_rest
    {
        public function __construct()
        {
            parent::__construct();

            //Used for testing
            $this->_has_testing_enabled = true;

            //Used to enable hover for the formatter
            $this->_enable_in_hover = true;
        }

        /**
         * test
         * Returns true or false to test criteria
         *
         * @return Boolean true if test completes successfully
         */
        public function test()
        {
            $item = $this->getItem(array('id'=>'1'));
            return !empty($item['firstname']) && ($item['firstname']
== 'John');
        }

        /**
         * getItem
         * Returns an array containing a key/value pair(s) of a source
record
         *
         * @param Array $args Array of arguments to search/filter by
         * @param String $module String optional value of the module
```

```
that the connector framework is attempting to map to
         * @return Array of key/value pair(s) of the source record;
empty Array if no results are found
         */
        public function getItem($args=array(), $module=null)
        {
            $result = null;
            if ($args['id'] == 1)
            {
                $result = array();
                $result['id'] = '1'; //Unique record identifier
                $result['firstname'] = 'John';
                $result['lastname'] = 'Doe';
                $result['email'] = 'john.doe@sugar.crm';
                $result['state'] = 'CA';
            }
            else if ($args['id'] == 2)
            {
                $result = array();
                $result['id'] = '2'; //Unique record identifier
                $result['firstname'] = 'Jane';
                $result['lastname'] = 'Doe';
                $result['email'] = 'jane.doe@sugar.crm';
                $result['state'] = 'HI';
            }

            return $result;
        }

        /**
         * getList
         * Returns a nested array containing a key/value pair(s) of a
source record
         *
         * @param Array $args Array of arguments to search/filter by
         * @param String $module String optional value of the module
that the connector framework is attempting to map to
         * @return Array of key/value pair(s) of source record; empty
Array if no results are found
         */
        public function getList($args=array(), $module=null)
        {
            $results = array();
            if(!empty($args['name']['last']) &&
strtolower($args['name']['last']) == 'doe')
            {
```

```
                $results[1] = array('
                    id' => 1,
                    'firstname' => 'John',
                    'lastname' => 'Doe',
                    'email' => 'john.doe@sugar.crm',
                    'state' => 'CA'
                );

                $results[2] = array(
                    'id' => 2,
                    'firstname' => 'Jane',
                    'lastname' => 'Doe',
                    'email' => 'john.doe@sugar.crm',
                    'state' => 'HI'
                );
            }

            return $results;
        }
    }

?>
```

Vardefs (vardefs.php)

The next step is to provide a list of variable definitions that your connector uses. These should be the fields used by your connector. For example, if your connector is a person lookup service, then these fields may be a first and last name, email address and phone number. You must also provide a unique field identifier for each connector record. This unique field needs to be named "id". Each vardef field entry is a defined within a PHP Array variable. The syntax is similar to a Sugar module's vardefs.php file with the exception of the 'hidden', 'input', 'search', 'hover' and 'options' keys.

- The 'hidden' key/value parameter is used to hide the connector's field in the framework. The required "id" field should be declared hidden because this is a unique record identifier that is used internally by the connector framework.
- The 'search' key/value parameter is an optional entry used to specify which connector field(s) are searchable. In step 1 of the connector wizard screen, a search form will be generated for your connector so that the user may optionally refine the list of results shown. Currently, we do not filter the fields that may be added to the search form so the use of the 'search' key/value parameter serves more as a visual indication.
- The 'hover' key/value parameter is an optional entry to support the hover

functionality. A vardef field that is denoted as the hover field contains the value the hover code will use in displaying a popup that displays additional detail in the Detail Views.
- The 'options' key/value parameter allows the developer to map values returned by the connector to values that may be used by the Sugar database.

In our example, the state field returns the abbreviated value of the State (CA for California, HI for Hawaii, etc.). If we wish to use the State name instead of the abbreviated name, we may specify the 'options' key/value parameter and then add the mapping entry to enable this translation. This is especially helpful should your system depend on a predefined set of values that differ from those returned by the connector. Here is a complete example of our example connector's vardefs.php file:

```
<?php

    if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    $dictionary['ext_rest_example'] = array(
        'comment' => 'vardefs for example connector',
        'fields' => array (
            'id' => array (
                'name' => 'id',
                'vname' => 'LBL_ID',
                'type' => 'id',
                'comment' => 'Unique identifier',
                'hidden' => true,
            ),
            'fullname' => array (
                'name' => 'fullname',
                'vname' => 'LBL_FULL_NAME',
                'hover' => true,
            ),
            'firstname' => array (
                'name' => 'firstname',
                'vname' => 'LBL_FIRST_NAME',
            ),
            'lastname' => array (
                'name' => 'lastname',
                'vname' => 'LBL_LAST_NAME',
                'input' => 'lastname',
                'search' => true,
            ),
            'email' => array (
                'name' => 'email',
```

```
                'vname' => 'LBL_EMAIL',
            ),
            'state' => array (
                'name' => 'state',
                'vname' => 'LBL_STATE',
                'options' => 'states_dom',
            ),
        )
    );

?>
```

Things to Note

The 'input' key/value parameter is an optional entry to provide an input argument name conversion. In other words, if your connector service expects a "firstName" argument but your connector's field is named "firstname", you may provide an additional input entry so that the connector framework will call your connector's getItem() and getList() methods with an argument named "firstName". Typically, the 'input' key/value parameter is used to support searching.

```
'lastname' => array (
     'name' => 'lastname',
     'vname' => 'LBL_LAST_NAME',
     'input' => 'lastName',
     'search' => true,
     'hover' => 'true',
),
```

Although the benefit of this is probably not captured well in this example, you could potentially have a service that groups arguments in a nested array. For example imagine the following array of arguments for a connector service:

```
$args['name']['first']
$args['name']['last']
$args['phone']['mobile']
$args['phone']['office']
```

Here we have an array with 'name' and 'phone' indexes in the first level. If your connector expects arguments in this format then you may supply an input key/value entry in your vardefs.php file to do this conversion. The input key value should be delimited with a period (.).

```
'lastname' => array (
     'name' => 'lastname',
```

```
     'vname' => 'LBL_LAST_NAME',
     'input' => 'name.last', // Creates Array argument
['name']['last']
     'search' => true,
     'hover' => 'true',
),
```

Configuration (config.php)

The configuration file (config.php) holds a PHP array with two keys. The "name" is used to provide a naming label for your connector that will appear in the tabs throughout the application. The "properties" key may be used to store runtime properties for your connector. Here we have simply provided the name "Example" and a properties value that we may use to control the various aspects of our connector such as max results.

./custom/modules/Connectors/connectors/sources/ext/rest/example/config.php

```
<?php
/***CONNECTOR SOURCE***/
$config['name'] = 'Example';
$config['properties']['max_results'] = 50;
```

Labels (.lang.php)

The next step is to create a language file with labels for your application. Notice that the properties defined in the config.php file are indexed by the property key ("max_results"). Otherwise, the vardefs.php entries should be indexed off the "vname" values.

./custom/modules/Connectors/connectors/sources/ext/rest/example/language/en_us.lang.php.

```
<?php

    if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    $connector_strings = array (
        //Vardef labels
        'LBL_LICENSING_INFO' => '<table border="0"
cellspacing="1"><tr><td valign="top" width="35%"
class="dataLabel">Licensing Info</td></tr></table>',
        'LBL_FULL_NAME' => 'Full Name',
```

```
        'LBL_FIRST_NAME' => 'First Name',
        'LBL_LAST_NAME' => 'Last Name',
        'LBL_EMAIL' => 'Email Address',
        'LBL_STATE' => 'State',

        //Configuration labels
        'max_results' => 'Max Results',
    );

?>
```

Mappings (mapping.php)

An optional mapping.php file may be provided so that default mappings are defined. These mappings assist the connector framework's component class. In our vardefs.php file, we enabled the hover link for the full_name field. To explicitly place this hover link on the full_name field for the Contacts module we provide the mapping entry as follows. A mapping.php file is needed though if you use the 'options' attribute for entries in the vardefs.php file.

```
<?php

$mapping = array (
  'beans' =>
  array (
    'Contacts' =>
    array (
      'firstname' => '',
      'lastname' => '',
      'email' => '',
      'state' => '',
      'fullname' => '',
      'id' => '',
    ),
  ),
);
```

Note that you can also predefine the mapping for the user as shown below:

```
<?php

$mapping = array (
  'beans' =>
  array (
    'Contacts' =>
```

```
    array (
      'firstname' => 'first_name',
      'lastname' => 'last_name',
      'email' => 'email1',
      'state' => 'primary_address_state',
      'fullname' => 'full_name',
      'id' => 'id',
    ),
  ),
);
```

Formatter

The optional formatter components are used by the connector framework to render a widget that may display additional details and information. Currently, they are shown in the detail view screens for modules that are enabled for the connector. You should note that formatters should reside in a directory structure similar to their source connectors involving the protocol type. An example of this structure is ./custom/modules/Connectors/connectors/formatters/ext/<protocol type>/<connector name>/

The first step in creating a formatter is to enable hover in its source class. For this example, we will need to make sure that the contruct of our source class (./custom/modules/Connectors/connectors/sources/ext/rest/example/example.php) has _enable_in_hover set to true:

```
$this->_enable_in_hover = true;
```

Next, we will create our formatter class. Our formatter class, ext_rest_example_formatter, will handle how the hover widget is displayed.

./custom/modules/Connectors/connectors/formatters/ext/rest/example/example.php

```
<?php

    if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');


require_once('include/connectors/formatters/default/formatter.php');

    class ext_rest_example_formatter extends default_formatter
    {

        public function getDetailViewFormat()
```

```
        {
           $mapping = $this->getSourceMapping();
           $mapping_name =
!empty($mapping['beans'][$this->_module]['fullname']) ?
$mapping['beans'][$this->_module]['fullname'] : '';

           if(!empty($mapping_name))
           {
               $this->_ss->assign('mapping_name', $mapping_name);
               return $this->fetchSmarty();
           }


$GLOBALS['log']->error($GLOBALS['app_strings']['ERR_MISSING_MAPPING_EN
TRY_FORM_MODULE']);
           return '';
        }

        public function getIconFilePath()
        {
           //icon for display
           return 'themes/Sugar/images/icon_basic_32.png';
        }

    }

?>
```

Next we will define our display template.

./custom/modules/Connectors/connectors/formatters/ext/rest/example/tpls/display.tpl

```
<script type="text/javascript" src="{sugar_getjspath
file='include/connectors/formatters/default/company_detail.js'}"></scr
ipt>

{literal}
<style type="text/css">
    .yui-panel .hd {
        background-color:#3D77CB;
        border-color:#FFFFFF #FFFFFF #000000;
        border-style:solid;
        border-width:1px;
        color:#000000;
        font-size:100%;
```

```
        font-weight:bold;
        line-height:100%;
        padding:4px;
        white-space:nowrap;
    }
</style>
{/literal}

<script type="text/javascript">
    function show_ext_rest_example(event)
    {literal}
    {
        var xCoordinate = event.clientX;
        var yCoordinate = event.clientY;
        var isIE = document.all?true:false;

        if(isIE) {
            xCoordinate = xCoordinate + document.body.scrollLeft;
            yCoordinate = yCoordinate + document.body.scrollTop;
        }

    {/literal}

        cd = new CompanyDetailsDialog("example_popup_div", '<div
id="example_div">Connector Content</div>', xCoordinate, yCoordinate);
        cd.setHeader("{$fields.{{$mapping_name}}.value}");
        cd.display();

    {literal}
    }
    {/literal}
</script>
```

Building a Module Loadable Package

The final step of a connector would be to zip your connector's contents along with a manifest.php file so that it may be installed by the Module Loader.

<basepath>/manifest.php

```
<?php

     $manifest = array (
         'acceptable_sugar_flavors' => array (),
         'acceptable_sugar_versions' => array(),
```

```
        'is_uninstallable' => true,
        'name' => 'Example Connector',
        'description' => 'A connector example',
        'author' => 'SugarCRM',
        'published_date' => '2013/02/12',
        'version' => '1',
        'type' => 'module',
        'icon' => '',
    );

    $installdefs = array (
        'id' => 'ext_rest_example',
        'connectors' => array (
            array (
                'connector' => '<basepath>/example/source',
                'formatter' => '<basepath>/example/formatter',
                'name' => 'ext_rest_example',
            ),
        ),
    );
?>
```

This manifest will install the specified files in the paths for 'connector' and 'formatter' as a connector. The path for the connector is determined from the connector 'name' index. In this case, with a name of ext_rest_example, the sources will be installed to ./custom/modules/Connectors/connectors/sources/ext/rest/example/ and the formatters to ./custom/modules/Connectors/connectors/formatters/ext/rest/example/.

A full module loadable example is available for download here.

Last Modified: 01/15/2016 10:58am

Dashlets

Overview of dashlets.

Last Modified: 11/17/2015 11:37pm

Introduction

Sugar Dashlets

Dashlets use the abstract factory design pattern. Individual dashlets extend the base abstract class Dashlet.php, list view Dashlets extend the base abstract class DashletGeneric.php, and chart dashlets extend the base abstract class DashletGenericChart.php.

Sugar Dashlet instances must be stored in one of the following directories:

- ./modules/<module>/Dashlets/
- ./custom/modules/<module>/Dashlets/

Sugar Dashlet developers use the ./custom/ directory to ensure their Sugar Dashlets are not manipulated by upgrades. Sugar Dashlets offered in base Sugar releases are located in the standard ./modules/ directory.

Sugar Dashlet Files

The file name containing the main Sugar Dashlet code must match the Sugar Dashlet's class name. For example, the Sugar Dashlet class JotPadDashlet is located in the file ./modules/Home/Dashlets/JotPadDashlet/JotPadDashlet.php. The JotPadDashlet is a sample Sugar Dashlet released originally in Sugar 4.5. It serves as a useful example from which to begin your development efforts.

A metadata file accompanies each Sugar Dashlet. It contains descriptive information about the Sugar Dashlet defined below:

```
$DashletMeta['JotPadDashlet'] = array (
    'title' => 'LBL_TITLE',
    'description' => 'LBL_TITLE',
    'icon' => 'themes/Sugar/images/Accounts.gif',
    'category' => 'Tools',
);
```

The naming convention for the metadata file is className.meta.php, where className is the name of the Sugar Dashlet. The metadata file must reside in the same directory as the Sugar Dashlet code. For JotPad Dashlet, the meta file is stored in ./modules/Home/Dashlets/JotPadDashlet/JotPadDashlet.meta.php.

The 'title' and 'description' elements are translated. If the values here match a key in the array $DashletStrings (from the language file) then they will be translated, otherwise it will display the literal string. (It is a best practice to use translatable

language strings so that your Sugar Dashlet is international!)

Language files have a similar naming convention: className.locale.lang.php (for example ./modules/Home/Dashlets/JotPadDashlet/JotpadDashlet.en_us.lang.php)

Icon files can either be defined in the .metadata file or included in the Sugar Dashlet Directory (for example ./modules/Home/Dashlets/JotPadDashlet/JotPadDashlet.icon.png).

Sugar scans for image files in the corresponding Sugar Dashlet directory.

Templating

The suggested templating engine for Sugar Dashlets is Smarty. This is not a requirement.

Categories

There are five categories for Sugar Dashlets:

- Module Views : Generic views of data in modules
- Portal : Sugar Dashlets that allow access to outside data (RSS, Web services, etc)
- Charts : Data charts
- Tools : Various tools such as notepad, calculator, or world clock
- Miscellaneous : Any other Sugar Dashlet

Sugar Dashlet base class

The main Sugar Dashlet base class is ./include/Dashlets/Dashlet.php and all Sugar Dashlets should extend this class.

Assign a unique ID to each Sugar Dashlet. This ID is used in the displayed HTML document and enable multiple Sugar dashlets of the same type to be included on the page.

Sugar Dashlets are stored in the table user_preferences under the Dashlet's name and home category.

The options element stores the options for the Sugar dashlet. This element is loaded/stored by storeOptions/loadOptions functions in the base Dashlet class.

Sugar Dashlets JavaScript

Sugar Dashlet utility functions are located in ./include/JavaScript/Dashlets.js and contains the following:

```
postForm: function(theForm, callback) {}
```

postForm is used to post the configuration form through AJAX.

The callback to remove the configuration dialog is SUGAR.sugarHome.uncoverPage:

```
callMethod: function(DashletId, methodName, postData, refreshAfter,
callback) {}
```

callMethod is a generic way to call a method in a Dashlet class. Use this function to generically call a method within your Dashlet class (php side). Refresh your Dashlet after a call and utilize a callback function (optional). This method can also be used to proxy AJAX calls to Web services that do not exist in the Sugar installation, for example, Google Maps Mash-up.

Last Modified: 01/15/2016 10:05pm

Custom Dashlets

Overview

Creating custom dashlets.

Sugar Dashlets

A Module View is the simplest Sugar Dashlet to create. This is a customizable ListView of a Sugar Dashlet. For this section we will use the MyAccountsDashlet as an example.
./modules/Accounts/Dashlets/MyAccountsDashlet/MyAccountsDashlet.php

```
<?php

if (!defined('sugarEntry') || !sugarEntry)
    die('Not A Valid Entry Point');

require_once('include/Dashlets/DashletGeneric.php');
```

```
class MyAccountsDashlet extends DashletGeneric
{
    function MyAccountsDashlet($id, $def = null)
    {
        global $current_user, $app_strings;

require('modules/Accounts/Dashlets/MyAccountsDashlet/MyAccountsDashlet
.data.php');

        parent::DashletGeneric($id, $def);

        if (empty($def['title']))
            $this->title = translate('LBL_HOMEPAGE_TITLE',
'Accounts');

        $this->searchFields =
$dashletData['MyAccountsDashlet']['searchFields'];
        $this->columns      =
$dashletData['MyAccountsDashlet']['columns'];

        $this->seedBean = new Account();
    }

    /**
     * Overrides the generic process to include custom logic for email
addresses,
     * since they are no longer stored in  a list view friendly
manner.
     * (A record may have an undetermined number of email addresses).
     *
     * @param array $lvsParams
     */

    function process($lvsParams = array())
    {
        if (isset($this->displayColumns) && array_search('email1',
$this->displayColumns) !== false) {
            $lvsParams['custom_select'] = ', email_address as email1';
            $lvsParams['custom_from']   = ' LEFT JOIN
email_addr_bean_rel eabr ON eabr.deleted = 0 AND bean_module =
\'Accounts\'' . ' AND eabr.bean_id = accounts.id AND primary_address =
1' . ' LEFT JOIN email_addresses ea ON ea.deleted = 0 AND ea.id =
eabr.email_address_id';
        }
```

```
        if (isset($this->displayColumns) &&
array_search('parent_name', $this->displayColumns) !== false) {
            $lvsParams['custom_select'] =
empty($lvsParams['custom_select']) ? ', a1.name as parent_name ' :
$lvsParams['custom_select'] . ', a1.name as parent_name ';
            $lvsParams['custom_from']   =
empty($lvsParams['custom_from']) ? ' LEFT JOIN accounts a1 on a1.id =
accounts.parent_id' : $lvsParams['custom_from'] . ' LEFT JOIN accounts
a1 on a1.id = accounts.parent_id';
        }

        parent::process($lvsParams);
    }
}

?>
```

All the metadata for this Sugar Dashlet is defined in the constructor. $searchFields are the search inputs that can be applied to the view. Defining these here will tell which input fields to generate corresponding filters when the user configures the Sugar Dashlet. $columns define the available columns to the user. These contain the visible columns and the columns the user can make visible. Initially, both columns and searchFields are defined in MyAccountsDashlet.data.php.

./modules/Accounts/Dashlets/MyAccountsDashlet/MyAccountsDashlet.data.php

```
<?php

if (!defined('sugarEntry') || !sugarEntry)
    die('Not A Valid Entry Point');

global $current_user;

$dashletData['MyAccountsDashlet']['searchFields'] = array(
    'date_entered' => array(
        'default' => ''
    ),
    'account_type' => array(
        'default' => ''
    ),
    'industry' => array(
        'default' => ''
    ),
    'billing_address_country' => array(
        'default' => ''
    ),
    'team_id' => array(
        'default' => '',
```

```
        'label' => 'LBL_TEAMS'
    ),
    'assigned_user_id' => array(
        'type' => 'assigned_user_name',
        'default' => $current_user->name,
        'label' => 'LBL_ASSIGNED_TO'
    )
);
$dashletData['MyAccountsDashlet']['columns']      = array(
    'name' => array(
        'width' => '40',
        'label' => 'LBL_LIST_ACCOUNT_NAME',
        'link' => true,
        'default' => true
    ),
    'website' => array(
        'width' => '8',
        'label' => 'LBL_WEBSITE',
        'default' => true
    ),
    'phone_office' => array(
        'width' => '15',
        'label' => 'LBL_LIST_PHONE',
        'default' => true
    ),
    'phone_fax' => array(
        'width' => '8',
        'label' => 'LBL_PHONE_FAX'
    ),
    'phone_alternate' => array(
        'width' => '8',
        'label' => 'LBL_OTHER_PHONE'
    ),
    'billing_address_city' => array(
        'width' => '8',
        'label' => 'LBL_BILLING_ADDRESS_CITY'
    ),
    'billing_address_street' => array(
        'width' => '8',
        'label' => 'LBL_BILLING_ADDRESS_STREET'
    ),
    'billing_address_state' => array(
        'width' => '8',
        'label' => 'LBL_BILLING_ADDRESS_STATE'
    ),
    'billing_address_postalcode' => array(
```

```
        'width' => '8',
        'label' => 'LBL_BILLING_ADDRESS_POSTALCODE'
    ),
    'billing_address_country' => array(
        'width' => '8',
        'label' => 'LBL_BILLING_ADDRESS_COUNTRY',
        'default' => true
    ),
    'shipping_address_city' => array(
        'width' => '8',
        'label' => 'LBL_SHIPPING_ADDRESS_CITY'
    ),
    'shipping_address_street' => array(
        'width' => '8',
        'label' => 'LBL_SHIPPING_ADDRESS_STREET'
    ),
    'shipping_address_state' => array(
        'width' => '8',
        'label' => 'LBL_SHIPPING_ADDRESS_STATE'
    ),
    'shipping_address_postalcode' => array(
        'width' => '8',
        'label' => 'LBL_SHIPPING_ADDRESS_POSTALCODE'
    ),
    'shipping_address_country' => array(
        'width' => '8',
        'label' => 'LBL_SHIPPING_ADDRESS_COUNTRY'
    ),
    'email1' => array(
        'width' => '8',
        'label' => 'LBL_EMAIL_ADDRESS_PRIMARY'
    ),
    'parent_name' => array(
        'width' => '15',
        'label' => 'LBL_MEMBER_OF',
        'sortable' => false
    ),
    'date_entered' => array(
        'width' => '15',
        'label' => 'LBL_DATE_ENTERED'
    ),
    'date_modified' => array(
        'width' => '15',
        'label' => 'LBL_DATE_MODIFIED'
    ),
    'created_by_name' => array(
```

```
        'width' => '8',
        'label' => 'LBL_CREATED'
    ),
    'assigned_user_name' => array(
        'width' => '8',
        'label' => 'LBL_LIST_ASSIGNED_USER'
    ),
    'team_name' => array(
        'width' => '15',
        'label' => 'LBL_LIST_TEAM'
    )
);

?>
```

Please note that modifications made in studio to columns and searchFields will be located in ./custom/modules/Accounts/metadata/dashletviewdefs.php. These settings will overrdie the layout for all dashlets under that module name.
This data file along with the MyAccountsDashlet.meta.php file will create a generic module view Sugar Dashlet.
./Accounts/Dashlets/MyAccountsDashlet/MyAccountsDashlet.meta.php

```
<?php

if (!defined('sugarEntry') || !sugarEntry)
    die('Not A Valid Entry Point');

global $app_strings;

$dashletMeta['MyAccountsDashlet'] = array(
    'module' => 'Accounts',
    'title' => translate('LBL_HOMEPAGE_TITLE', 'Accounts'),
    'description' => 'A customizable view into Accounts',
    'category' => 'Module Views'
);

?>
```

Custom Sugar Dashlets

Sugar Dashlets are more than generic module views. They can provide unlimited functionality and integration.
For this section we will use the JotPad Sugar Dashlet as an example. The JotPad is a simple note taking Sugar Dashlet. A user double clicks on the Sugar Dashlet and can enter any text in the Sugar Dashlet. When the user clicks outside of the textarea, the text is automatically saved via AJAX.
There are six files that define this Sugar Dashlet residing in the

./modules/Home/Dashlets/JotPadDashlet/ directory:

1. JotPadDashlet.php – JotPad Class
2. JotPadDashlet.meta.php – metadata about the Sugar Dashlet
3. JotPadDashlet.tpl – Display Template
4. JotPadDashletOptions.tpl – Configuration template
5. JotPadDashletScript.tpl - Javascript
6. JotPadDashlet.en_us.lang.php – English Language file

JotPadDashlet.php

The JotPadDashlet.php file handles the display of the dashlet.

./modules/Home/Dashlets/JotPadDashlet/JotPadDashlet.php:

```
<?php

if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

require_once('include/Dashlets/Dashlet.php');

class JotPadDashlet extends Dashlet {
    var $savedText; // users's saved text
    var $height = '200'; // height of the pad

    /**
     * Constructor
     *
     * @global string current language
     * @param guid $id id for the current dashlet (assigned from Home
module)
     * @param array $def options saved for this dashlet
     */
    function JotPadDashlet($id, $def) {
        $this->loadLanguage('JotPadDashlet'); // load the language
strings here

        if(!empty($def['savedText']))  // load default text is none is
defined
            $this->savedText = $def['savedText'];
        else
            $this->savedText =
$this->dashletStrings['LBL_DEFAULT_TEXT'];

        if(!empty($def['height'])) // set a default height if none is
```

```
set
            $this->height = $def['height'];

        parent::Dashlet($id); // call parent constructor

        $this->isConfigurable = true; // dashlet is configurable
        $this->hasScript = true;  // dashlet has javascript attached
to it

        // if no custom title, use default
        if(empty($def['title'])) $this->title =
$this->dashletStrings['LBL_TITLE'];
        else $this->title = $def['title'];
    }

    /**
     * Displays the dashlet
     *
     * @return string html to display dashlet
     */
    function display() {
        $ss = new Sugar_Smarty();
        $ss->assign('savedText',
SugarCleaner::cleanHtml($this->savedText));
        $ss->assign('saving', $this->dashletStrings['LBL_SAVING']);
        $ss->assign('saved', $this->dashletStrings['LBL_SAVED']);
        $ss->assign('id', $this->id);
        $ss->assign('height', $this->height);

        $str =
$ss->fetch('modules/Home/Dashlets/JotPadDashlet/JotPadDashlet.tpl');
        return
parent::display($this->dashletStrings['LBL_DBLCLICK_HELP']) . $str . '

'; // return parent::display for title and such
    }

    /**
     * Displays the javascript for the dashlet
     *
     * @return string javascript to use with this dashlet
     */
    function displayScript() {
        $ss = new Sugar_Smarty();
        $ss->assign('saving', $this->dashletStrings['LBL_SAVING']);
        $ss->assign('saved', $this->dashletStrings['LBL_SAVED']);
```

```
        $ss->assign('id', $this->id);

        $str =
$ss->fetch('modules/Home/Dashlets/JotPadDashlet/JotPadDashletScript.tp
l');
        return $str; // return parent::display for title and such
    }

    /**
     * Displays the configuration form for the dashlet
     *
     * @return string html to display form
     */
    function displayOptions() {
        global $app_strings;

        $ss = new Sugar_Smarty();
        $ss->assign('titleLbl',
$this->dashletStrings['LBL_CONFIGURE_TITLE']);
        $ss->assign('heightLbl',
$this->dashletStrings['LBL_CONFIGURE_HEIGHT']);
        $ss->assign('saveLbl', $app_strings['LBL_SAVE_BUTTON_LABEL']);
        $ss->assign('clearLbl',
$app_strings['LBL_CLEAR_BUTTON_LABEL']);
        $ss->assign('title', $this->title);
        $ss->assign('height', $this->height);
        $ss->assign('id', $this->id);

        return parent::displayOptions() .
$ss->fetch('modules/Home/Dashlets/JotPadDashlet/JotPadDashletOptions.t
pl');
    }

    /**
     * called to filter out $_REQUEST object when the user submits the
configure dropdown
     *
     * @param array $req $_REQUEST
     * @return array filtered options to save
     */
    function saveOptions($req) {
        global $sugar_config, $timedate, $current_user, $theme;
        $options = array();
        $options['title'] = $_REQUEST['title'];
        if(is_numeric($_REQUEST['height'])) {
            if($_REQUEST['height'] > 0 && $_REQUEST['height'] <= 300)
```

```
$options['height'] = $_REQUEST['height'];
            elseif($_REQUEST['height'] > 300) $options['height'] =
'300';
            else $options['height'] = '100';
        }

        $options['savedText'] = $this->savedText;
        return $options;
    }

    /**
     * Used to save text on textarea blur. Accessed via
Home/CallMethodDashlet.php
     * This is an example of how to to call a custom method via ajax
     */
    function saveText() {
        $json = getJSONobj();
     if(isset($_REQUEST['savedText'])) {
            $optionsArray = $this->loadOptions();

$optionsArray['savedText']=$json->decode(html_entity_decode($_REQUEST[
'savedText']));

$optionsArray['savedText']=SugarCleaner::cleanHtml(nl2br($optionsArray
['savedText']));
            $this->storeOptions($optionsArray);

        }
        else {
            $optionsArray['savedText'] = '';
        }
        echo 'result = ' . $json->encode(array('id' =>
$_REQUEST['id'],
                                        'savedText' =>
$optionsArray['savedText']));
    }
}


?>
```

JotPadDashletOptions.tpl

The JotPadDashletOptions.tpl file handles the display of the dashlet options.

./modules/Home/Dashlets/JotPadDashlet/JotPadDashletOptions.tpl

```
<div style='width: 500px'>
<form name='configure_{$id}' action="index.php" method="post"
onSubmit='return SUGAR.dashlets.postForm("configure_{$id}",
SUGAR.mySugar.uncoverPage);'>
<input type='hidden' name='id' value='{$id}'>
<input type='hidden' name='module' value='Home'>
<input type='hidden' name='action' value='ConfigureDashlet'>
<input type='hidden' name='to_pdf' value='true'>
<input type='hidden' name='configure' value='true'>
<table width="400" cellpadding="0" cellspacing="0" border="0"
class="edit view" align="center">
<tr>
  <td valign='top' nowrap class='dataLabel'>{$titleLbl}</td>
  <td valign='top' class='dataField'>
    <input class="text" name="title" size='20' value='{$title}'>
  </td>
</tr>
<tr>
  <td valign='top' nowrap class='dataLabel'>{$heightLbl}</td>
  <td valign='top' class='dataField'>
    <input class="text" name="height" size='3' value='{$height}'>
  </td>
</tr>
<tr>
  <td align="right" colspan="2">
      <input type='submit' class='button' value='{$saveLbl}'>
    </td>
</tr>
</table>
</form>
</div>
```

The important thing to note here is the onSubmit. All configure forms should have this statement to uncover the page to remove the configuration dialog.
Note: It is important to separate your JavaScript into a separate JavaScript file. This is because Sugar Dashlets are dynamically added to a page through AJAX. The HTML included into JavaScript is not evaluated when dynamically included.
It is important that all JavaScript functions are included in this script file. Inline JavaScript (<a href onclick='' etc) will still function. If the Sugar Dashlet has JavaScript and a user dynamically adds it to the page, the Sugar Dashlet will not be accessible until after the user reloads the page.
Therefore it is beneficial to use as many generic methods in Dashlet.js as possible (Dashlets.callMethod() specifically!).

JotPadDashletScripts.tpl

The JotPadDashletScripts.tpl handles the generic javascript for the dashlet.

./modules/Home/Dashlets/JotPadDashlet/JotPadDashletScripts.tpl
```
{literal}

{/literal}
```

Refreshing the Sugar Dashlet Cache

To add a Sugar Dashlet to your SugarCRM installation, you can use the dashlets installdef in Module Loader to install your Sugar Dashlet Package to the ./custom/Home/Dashlets/ directory. If you are developing or need to copy the dashlet to a different modules directory, you will need to navigate to Admin > Repair > Rebuild Sugar Dashlets. This will rebuild the dashlet cache locatd in ./cache/dashlets/dashlets.php that maps the dashlets.

Packaging Generic Sugar Dashlets

If you are packaging a generic dashlet that is not module specific, you can use the dashlets installdef. This will install the dashlet to ./custom/modules/Home/Dashlets/<dashlet>/.
manifest.php
```
<?php

     $manifest    = array(
         'acceptable_sugar_flavors' => array(),
         'acceptable_sugar_versions' => array(),
         'author' => 'SugarCRM',
         'description' => 'Installs the dashlet using the dashlets
installdef',
         'icon' => '',
         'is_uninstallable' => true,
         'name' => 'Dashlet installer example',
         'published_date' => '2013-01-29 2013 13:49:58',
         'type' => 'module',
         'version' => '1.0'
     );

     $installdefs = array(
         'id' => 'package_1359467398',
         'dashlets' => array(
             0 => array( //The name to install the dashlet under
'name' => 'MyDashlet',
                 //This directory contains the dashlet files
```

```
                'from' => '<basepath>/MyDashlet'
            )
        )
    );

?>
```

If you are creating a module specific dashlet, you will have to move the dashlet to the directory using the copy installdef as shown below:

manifest.php

```
<?php

    $manifest    = array(
        'acceptable_sugar_flavors' => array(),
        'acceptable_sugar_versions' => array(),
        'author' => 'SugarCRM',
        'description' => 'Installs the dashlet using the copy
installdef',
        'icon' => '',
        'is_uninstallable' => true,
        'name' => 'Dashlet installer example',
        'published_date' => '2013-01-29 2013 13:49:58',
        'type' => 'module',
        'version' => '1.0'
    );

    $installdefs = array(
        'id' => 'package_1359467399',
        'copy' => array(
            0 => array(
                'from' => '/MyDashlet/',
                'to' => 'custom/modules/<module>/Dashlets/MyDashlet'
            )
        )
    );

?>
```

Please note that if you are installing a dashlet using the copy installdef, you will need to navigate to Admin > Repair > Rebuild Sugar Dashlets. This will rebuild the dashlet cache.

More information on the manifest file can be found in the Introduction to the Manifest section.

Creating Custom Chart Dashlets

Creating a custom chart dashlet is very similar to creating the MyAccountsDashlet described above. The main difference is that you will need to override the display() method in your class to build the chart, using the SugarChartFactory class included with SugarCRM. Beginning in Sugar 6.2, we have switched the charts to be rendered through JavaScript. The SugarChartFactory returns a subclass of SugarChart. See below for an example of display() method as used in the Outcome by Month dashlet.

./modules/Charts/Dashlets/OutcomeByMonthDashlet/OutcomeByMonthDashlet.php

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
require_once('include/Dashlets/DashletGenericChart.php');
class OutcomeByMonthDashlet extends DashletGenericChart
{
    public $obm_ids = array();
    public $obm_date_start;
    public $obm_date_end;

    /**
     * @see DashletGenericChart::$_seedName
     */
    protected $_seedName = 'Opportunities';
    /**
     * @see DashletGenericChart::__construct()
     */
    public function __construct(
        $id,
        array $options = null
        )
    {
        global $timedate;
 if(empty($options['obm_date_start']))
            $options['obm_date_start'] = $timedate->nowDbDate();
        if(empty($options['obm_date_end']))
            $options['obm_date_end'] =
$timedate->asDbDate($timedate->getNow()->modify("+6 months"));
        parent::__construct($id,$options);
    }
    /**
     * @see DashletGenericChart::displayOptions()
     */
    public function displayOptions()
    {
```

```
        if (!isset($this->obm_ids) || count($this->obm_ids) == 0)
 $this->_searchFields['obm_ids']['input_name0'] =
array_keys(get_user_array(false));
        return parent::displayOptions();
    }
    /**
     * @see DashletGenericChart::display()
     */
    public function display()
    {
        $currency_symbol =
$GLOBALS['sugar_config']['default_currency_symbol'];
        if ($GLOBALS['current_user']->getPreference('currency')){
            $currency = new Currency();

$currency->retrieve($GLOBALS['current_user']->getPreference('currency'
));
            $currency_symbol = $currency->symbol;
        }
        require("modules/Charts/chartdefs.php");
        $chartDef = $chartDefs['outcome_by_month'];
        require_once('include/SugarCharts/SugarChartFactory.php');
        $sugarChart = SugarChartFactory::getInstance();
        $sugarChart->setProperties('',
            translate('LBL_OPP_SIZE', 'Charts') . ' ' .
$currency_symbol . '1' .translate('LBL_OPP_THOUSANDS', 'Charts'),
            $chartDef['chartType']);
        $sugarChart->base_url = $chartDef['base_url'];
        $sugarChart->group_by = $chartDef['groupBy'];
        $sugarChart->url_params = array();
        $sugarChart->getData($this->constructQuery());
        $sugarChart->is_currency = true;
        $sugarChart->data_set =
$sugarChart->sortData($sugarChart->data_set, 'm', false,
'sales_stage', true, true);
        $xmlFile = $sugarChart->getXMLFileName($this->id);
        $sugarChart->saveXMLFile($xmlFile,
$sugarChart->generateXML());
        return $this->getTitle('<div align="center"></div>') .
            '<div align="center">' . $sugarChart->display($this->id,
$xmlFile, '100%', '480', false) . '</div>'.
$this->processAutoRefresh();
 }
    /**
     * @see DashletGenericChart::constructQuery()
     */
```

```
    protected function constructQuery()
    {
        $query = "SELECT sales_stage,".

db_convert('opportunities.date_closed','date_format',array("'%Y-%m'"),
array("'YYYY-MM'"))." as m, ".
            "sum(amount_usdollar/1000) as total, count(*) as opp_count
FROM opportunities ";
        $this->getSeedBean()->add_team_security_where_clause($query);
        $query .= " WHERE opportunities.date_closed >=
".db_convert("'".$this->obm_date_start."'",'date') .
                        " AND opportunities.date_closed <=
".db_convert("'".$this->obm_date_end."'",'date') .
                        " AND opportunities.deleted=0";
        if (count($this->obm_ids) > 0)
            $query .= " AND opportunities.assigned_user_id IN ('" .
implode("','",$this->obm_ids) . "')";
        $query .= " GROUP BY sales_stage,".

db_convert('opportunities.date_closed','date_format',array("'%Y-%m'"),
array("'YYYY-MM'")) .
                    " ORDER BY m";
        return $query;
    }
}
```

Last Modified: 01/08/2017 04:38pm

Databases

All five Sugar editions support the MySQL and Microsoft SQL Server databases. Sugar Enterprise and Sugar Ultimate also support the DB2 and Oracle databases. In general, Sugar uses only common database functionality, and the application logic is embedded in the PHP code. Sugar does not use database triggers or stored procedures. This design simplifies coding and testing across different database vendors. The only implementation difference across the various supported databases is column types.

Indexes

Indexes can be defined in the main or custom vardefs.php for module in an array under the key indices. See below for an example of defining several indices:

```
'indices' => array(
     array(
         'name' => 'idx_modulename_name',
         'type' => 'index',
         'fields' => array('name'),
     ),
     array(
         'name' => 'idx_modulename_assigned_deleted',
         'type' => 'index',
         'fields' => array('assigned_user_id', 'deleted'),
     ),
 ),
```

The name of the index must start with idx_ and must be unique across the database. Possible values for type include primary for a primary key or index for a normal index. The fields list matches the column names used in the database.

Primary Keys, Foreign Keys, and GUIDs

By default, Sugar uses globally unique identification values (GUIDs) for primary keys for all database records. Sugar provides a create_guid() utility function for creating these GUIDs in the following format: aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee. The primary key column length is 36 characters.

The GUID format and value has no special meaning (relevance) in Sugar other than the ability to match records in the database. Sugar links two records (such as an Accounts record with a Contacts record) with a specified ID in the record type relationship table (e.g. accounts_contacts).

Sugar allows a primary key to contain any unique string. This can be a different GUID algorithm, a key that has some meaning (such as bean type first, followed by info), an external key, and/or auto-incrementing numbers (converted to strings). Sugar chose GUIDs over auto-incrementing keys to allow for easier data synchronization across databases and avoid primary key collisions when one of the following occurs:

- Sugar Offline Client (part of Sugar Enterprise) syncs data with the main Sugar installation.
- Sugar SOAP APIs are used for data synchronization.
- Tools like Talend are used for data synchronization.

Offline Client uses GUIDs for primary keys for ease of implementation and simpler handling of data conflicts compared to other schemes. If a developer changes

Sugar to use some other ID scheme and needs to accommodate data synchronization across data stores, IDs need to be partitioned ahead of time or a system similar to the Sugar implementation for Cases, Quotes, and Bugs created. For modules like these that have human-readable ID numbers (integers) that need to be synchronized across databases, Sugar implements a server ID that is globally unique and concatenates it with an incrementing Case, Quotes or Bug number. Attempting such a change to Sugar requires some careful planning and implementation.

If data synchronization is not an issue, the primary key format can be changed to some other unique string.

You can also import data from a previous system with one primary key format and make all new records in Sugar use the GUID primary key format. All keys need to be stored as globally unique strings with no more than 36 characters. If multiple modules contain records with matching ids, you may experience undesired behaviors within the system.

To perform any of the following:

- Implement a new primary key method
- Import existing data with a different primary key format based on the existing GUID mechanism for new records

Make note of the following:

- Quote characters : Sugar expects primary keys to be string types and will format the SQL with quotes. If you change the primary key types to an integer type, SQL errors may occur since Sugar stores all ID values in quotes in the generated SQL. The database may be able to ignore this issue. MySQL running in Safe mode experiences issues, for instance.
- Case-sensitivity : IDs abc and ABC are treated the same in MySQL but represent different values in Oracle. When migrating data to Sugar, some CRM systems may use case sensitive strings as their IDs on export. If this is the case, and you are running MySQL, you need to run an algorithm on the data to make sure all of the IDs are unique. One simple algorithm is to MD5 the ids that they provide. A quick check will let you know if there is a problem. If you imported 80,000 leads and there are only 60,000 in the system, some may have been lost due to non-unique primary keys, as a result of case sensitivity.
- Sugar only tracks the first 36 characters in the primary key. Any replacement primary key will either require changing all of the ID columns with one of an appropriate size or to make sure you do not run into any truncation or padding issues. MySQL in some versions has had issues with Sugar where the

IDs were not matching because it was adding spaces to pad the row out to the full size. MySQL's handling of char and varchar padding has changed in some of the more recent versions. To protect against this, you will want to make sure the GUIDs are not padded with blanks in the DB.

Last Modified: 09/26/2015 04:20pm

Entry Points

Overview of entry points.

Last Modified: 11/17/2015 11:37pm

Introduction

Overview

Introduction to entry points

Introduction to Entry Points

All entry points into the Sugar application are pre-defined to ensure that proper security and authentication steps are applied consistently across the entire application. The main entry points to note are:

- ./cron.php - Used by the Windows Scheduler Service or the cron service on Linux and Unix for executing the Sugar Scheduler periodically
- ./index.php - Default entry point into the Sugar application
- ./install.php - Used for initial install of the Sugar application
- ./soap.php - Entry point for all v1 SOAP calls. This entry point is deprecated and should not be used for API development
- ./service/{api version}/soap.php - Entry point for v2 - v4_1 SOAP calls
- ./service/{api version}/rest.php - Entry point for for v2 - v4_1 REST calls

All stock entry points can be found in ./include/MVC/Controller/entry_point_registry.php.

Accessing Entry Points

Available entry points can be accessed using a URL pattern as follows:

```
http://{sugar url}/index.php?entryPoint={entry point name}
```

The entry point name will be the specified index in the $entry_point_registry array. Access to this entry point outside of sugar will be dependent on the auth parameter defined in the $entry_point_registry array as well. The following section will outline creating custom entry points.

Last Modified: 09/26/2015 04:20pm

Custom Entry Points

Overview

How to create custom entry points.

Creating a Custom Entry Point

As of 6.3.x, entry points can be created using the extension framework. The entry point extension directory is located at ./custom/Extension/application/Ext/EntryPointRegistry/. Entry point files found is this directory will be compiled into ./custom/application/Ext/EntryPointRegistry/entry_point_registry.ext.php after a Quick Repair and Rebuild. Prior to 6.3.x, an entry point was added by creating the file ./custom/include/MVC/Controller/entry_point_registry.php. This method of creating entry points is still compatible but is not recommended from a best practices standpoint.

Entry point registries contain two properties:

- file - The path to the entry point.
- auth - A Boolean value that determines whether or not the user must be authenticated in order to access the entry point.

```
$entry_point_registry['customEntryPoint'] = array(
    'file' => 'path/to/customEntryPoint.php',
    'auth' => true
);
```

Example

The first step is to create the actual entry point. This is where all of the logic for your entry point will be located. This file can be located anywhere you choose. For my example, I will create:

./custom/customEntryPoint.php

```
<?php

    if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    echo "Hello World!";
```

Next, we will need to create our extension in the application extensions. This will be located at:
./custom/Extension/application/Ext/EntryPointRegistry/customEntryPoint.php

```
<?php

    $entry_point_registry['customEntryPoint'] = array(
        'file' => 'custom/customEntryPoint.php',
        'auth' => true
    );
```

Finally, navigate to Admin > Repair > Quick Repair and Rebuild. The system will then generate the file
./custom/application/Ext/EntryPointRegistry/entry_point_registry.ext.php
containing your registry entry. We are now able to access our entry point by navigating to:

```
http://{sugar url}/index.php?entryPoint=customEntryPoint
```

Last Modified: 01/08/2017 04:38pm

File Caching

Overview

An Overview of how caching works within SugarCRM

What It Does

Much of the user interface is built dynamically using templates from metadata and language string files. Sugar implements a file caching mechanism to improve the performance of the system by reducing the number of static metadata and language files that need to be resolved at runtime. This directory stores the cached template and language string files.

Where is the Cache?

In a stock instance, the cache is located in the sugar root 'cache' directory. If you would like to move this directory to a new location, you can update the config parameter 'cache_dir' in your config.php or config_override.php to meet your needs.

Developer Mode

To prevent caching while developing, a developer may opt to turn on developer mode.

```
Admin > System Settings > Advanced > Developer Mode
```

This is especially helpful when you are directly altering templates, metadata, or language files. The system automatically refreshes the file cache. Make sure to deactivate developer mode after completing customizations because this mode degrades system performance.

Last Modified: 01/08/2017 04:38pm

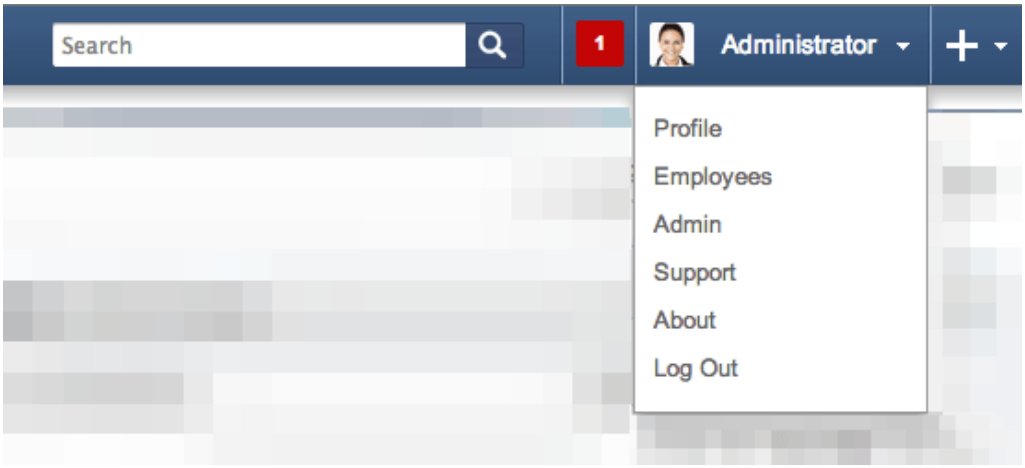
Global Control Links

Overview

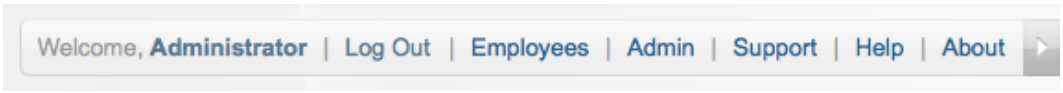
Managing global control links.

Link Location

Global control links are the links located in the profile drop down.



Sugar 6.0 through 6.4 will look similar to this:



Modifying the Links

As of 6.3.x, global control links can be created using the extension framework. The global links extension directory is located at ./custom/Extension/application/Ext/GlobalLinks/. Global link files found is this directory will be compiled into ./custom/application/Ext/GlobalLinks/links.ext.php after a Quick Repair and Rebuild.

Prior to 6.3.x, a global link was added by creating the file ./custom/include/globalControlLinks.php. This method of creating global links is still compatible but is not recommended from a best practices standpoint.

Examples

Adding a Link

This example will demonstrate how to add a link to the global links:

./custom/Extension/application/Ext/GlobalLinks/<file>.php

```
<?php

    $global_control_links['google'] = array(
```

```
        'linkinfo' => array(
            //String Text => URL
            $app_strings['LBL_SUGARCRM'] => 'http://www.sugarcrm.com'
        )
    );
```

Next, we will define the app_string:

./custom/Extension/application/Ext/Language/<language>.<file>.php

```
<?php

    $app_strings['LBL_SUGARCRM'] = 'SugarCRM';
```

Removing a Link

This example will demonstrate how to remove a global link:

./custom/Extension/application/Ext/GlobalLinks/<file>.php

```
<?php

    if (isset($global_control_links['employees']))
    {
        unset($global_control_links['employees']);
    }
```

Last Modified: 01/08/2017 04:38pm

Helper Classes

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/17/2015 11:37pm

Administration

Overview

The Administration class is used to manage settings stored in the database 'config' table.

The Administration Class

The Administration class is located in 'modules/Administration/Administration.php'. Settings modified using this class are written to the 'config' table..

Creating / Updating Settings

To create or update a specific setting, you can specify the new value using the Administraton 'saveSetting' function as shown below:

```
require_once('modules/Administration/Administration.php')

$administrationObj = new Administration();

//save the setting
$administrationObj->saveSetting("MyCategory", "MySetting",
'MySettingsValue');
```

Retrieving Settings

You can access the config settings by using the Administration 'retrieveSettings' function. You can filter the settings by category by passing in filter a parameter. If no value is passed to 'retrieveSettings', all settings will be returned. An example is show below:

```
require_once('modules/Administration/Administration.php');

$administrationObj = new Administration();

//Retrieve all settings in the category of 'MyCategory'.
//This parameter can be left empty to retrieve all settings.
$administrationObj->retrieveSettings('MyCategory');

//Use a specific setting
$MySetting = $administrationObj->settings['MyCategory_MySetting'];
```

Considerations

When looking to store custom settings, the administration class will store the settings in the 'config' table. Alternatively, you can use the Configurator class located in ./modules/Configurator/Configurator.php to store the settings in the 'config_override.php' file.

Last Modified: 09/26/2015 04:20pm

BeanFactory

Overview

An overview of the BeanFactory class which is used to retrieve bean objects.

The BeanFactory Class

The BeanFactory class, located in ./data/BeanFactory.php, is used for loading an instance of a SugarBean. This class should be used any time you are creating or retrieving bean objects. It will automatically handle any classes required for the bean.

Creating a SugarBean Object

newBean()

To create a new empty SugarBean, you can use the newBean() method. This method is typically used when creating a new record for a module or to call properties of the modules bean object.

```
$bean = BeanFactory::newBean($module);
```

newBeanByName()

Used to fetch a bean by its beanList name.

```
$bean = BeanFactory::newBeanByName($name);
```

Retrieving a SugarBean Object

getBean()

The getBean() method can be used to retrieve a specific record from the database. If a record id is not passed, a new bean object will be created.

```
$bean = BeanFactory::getBean($module, $record_id);
```

retrieveBean()

The retrieveBean() method can also be used to retrieve a specific record from the database. The difference between this method and getBean() is that null will be returned instead of an empty bean object if the retrieve fails.

```
$bean = BeanFactory::retrieveBean($module, $record_id);
```

Retrieving Module Keys

getObjectName()

The getObjectName() method will return the object name / dictionary key for a given module. This is normally the same as the bean name, but may not be for some modules such as Cases which has a key of 'aCase' and an name of 'Case'.

```
$moduleKey = BeanFactory::getObjectName($moduleName);
```

getBeanName()

The getBeanName() method will retrieve the bean class name given a module name.

```
$moduleClass = BeanFactory::getBeanName($module);
```

Last Modified: 09/26/2015 04:20pm

Configurator

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/17/2015 11:37pm

Core Settings

Overview

Sugar configuration settings.

Settings Architecture

When you first install Sugar, all of the default settings are located in ./config.php. As you begin configuring the system, the modified settings are stored in ./config_override.php. Settings in ./config.php are overridden by the values in ./config_override.php.

Settings

addAjaxBannedModules (Deprecated in future release)

Description	Disables a module from being loaded using the AJAX UI. This is generally used if you are experiencing display issues with modules and want to prevent it from being loaded by the AJAX UI.
Type	Array : Module list
Versions	6.3.0 - 6.7.9
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['addAjaxBannedModules'][] = 'Accounts';`

admin_access_control

Description	Removes Sugar Updates, Upgrade Wizard, Backups and Module Builder

	from the Admin menu. For developers, these restrictions can be found in ./include/MVC/Controller/file_access_control_map.php and overriden by creating ./custom/include/MVC/Controller/file_access_control_map.php.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['admin_access_control'] = true;`

admin_export_only

Description	Allow only users with administrative privileges to export data.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['admin_export_only'] = true;`

allow_pop_inbound

Description	Inbound email accounts are setup to work with IMAP protocols by default. If your email provider required POP3 access instead of IMAP, you can enables POP3 as an available inbound email

	protocol. Please note that mailboxes configured with a POP3 connection are not supported by SugarCRM and may cause unintended consequences. IMAP is the recommended protocol to use for inbound email accounts.
Type	Boolean
Range of values	true and false
Versions	5.5.1+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['allow_pop_inbound'] = true;`

allow_sendmail_outbound

Description	Enables the option of choosing sendmail as an SMTP server in Admin > Email Settings. Sendmail must be enabled on the server for this option to work. Please note that mailboxes configured with sendmail are not supported and may cause unintended consequences. Instances running on the On-Demand environment will have this setting enforced as false.
Type	Boolean
Range of values	true and false
Versions	5.5.1+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
On-Demand Value	false
Override Example	`$sugar_config['allow_sendmail_outbound'] = true;`

aws

Description	The Amazon AWS configuration. Used for storing uploads on S3. The upload_wrapper_class setting must also be updated to SugarUploadS3 to enable this configuration. You can do this by placing $sugar_config['upload_wrapper_class'] = 'SugarUploadS3'; in your config_override.php.
Type	Array
Versions	6.7.0+
Editions	Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['aws'] = array();`

aws.aws_key

Description	Amazon AWS public key.
Type	String : Key
Versions	6.7.0+
Editions	Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['aws']['aws_key'] = 'key';`

aws.aws_secret

Description	Amazon AWS secret key.
Type	String : Key
Versions	6.7.0+
Editions	Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['aws']['aws_secret'] = 'secret';`

aws.upload_bucket

Description	The Amazon S3 bucket name for uploads.
Type	String : S3 bucket name
Versions	6.7.0+
Editions	Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['aws']['upload_bucket'] = 'bucket';`

cache_dir

Description	This is the directory SugarCRM will store all cached files. Can be relative to Sugar root directory.
Type	String : Directory
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	cache/
Override Example	`$sugar_config['cache_dir'] = 'cache/';`

cache_expire_timeout

Description	The length of time cached items should be expired after.
Type	Integer : Seconds
Versions	6.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	300

Override Example	`$sugar_config['cache_expire_timeout'] = 400;`

calendar

Description	An array that defines all of the various settings for the Calendar module.
Type	Array
Versions	6.4.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['calendar'] = array();`

calendar.day_timestep

Description	Sets the default day time step.
Type	Integer : Days
Range of values	15, 30 and 60
Versions	6.4.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	15
Override Example	`$sugar_config['calendar']['day_timestep'] = 15;`

calendar.default_view

Description	Changes the default view in the calendar module.
Type	String
Range of values	'day', 'week', 'month' and 'share'
Versions	6.4.0+

Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['calendar']['default_view'] = 'week';`

calendar.items_draggable

Description	Enable/Disable drag-and-drop feature to move calendar items.
Type	Boolean
Range of values	true and false
Versions	6.4.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	true
Override Example	`$sugar_config['calendar']['items_draggable'] = true;`

calendar.items_resizable

Description	Sets whether items on the calendar can be resized via clicking and dragging.
Type	Boolean
Range of values	true and false
Versions	6.5.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	true
Override Example	`$sugar_config['calendar']['items_resizable'] = true;`

calendar.show_calls_by_default

Description	Display/Hide calls by default.

Type	Boolean
Range of values	true and false
Versions	6.4.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	true
Override Example	`$sugar_config['calendar']['show_calls_by_default'] = true;`

calendar.week_timestep

Description	The default week step size when viewing the calendar.
Type	Integer : Calendar Step Size
Range of values	15, 30, and 60
Versions	6.4.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['calendar']['week_timestep'] = 30;`

check_query

Description	Validates queries when adding limits.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	true
Override Example	`$sugar_config['check_query'] = true;`

check_query_cost

Description	Sets the maximum cost limit of a query.
Type	Integer
Versions	5.2.0+
Editions	Enterprise, Ultimate
Default Value	10
Override Example	`$sugar_config['check_query_cost'] = 10;`

cron

Description	Array that defines all of the cron parameters.
Type	Array
Versions	6.5.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['cron'] = aray();`

cron.max_cron_jobs

Description	Maximum jobs per cron run. Default is 10. If you are using a version prior to 6.5.14, you will need to also populate max_jobs to set this value due to bug #62936 (https://web.sugarcrm.com/support/issues/62936).
Type	Integer
Versions	6.5.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	6.5.x: 10 7.6.x: 25

Override Example	`$sugar_config['cron']['max_cron_jobs'] = 10;`

cron.max_cron_runtime

Description	Determines the maximum time in seconds that a single job should be allowed to run. If a single job exceeds this limit, cron.php is aborted with the long-running job marked as in progress in the job queue. The next time cron runs, it will skip the job that overran the limit and start on the next job in the queue. This limit is only enforced when cron.enforce_runtime is set to true.
Type	Integer : Seconds
Versions	6.5.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	180
Override Example	`$sugar_config['cron']['max_cron_runtime'] = 60;`

cron.min_cron_interval

Description	Minimum time between cron runs. Setting this to 0 will disable throttling completely.
Type	Integer : Seconds
Versions	6.5.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	30
Override Example	`$sugar_config['cron']['min_cron_interval'] = 30;`

custom_help_url

Description	Designate the URL used to redirect the user to help documentation.
Type	String : Website address
Versions	6.4.3+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	http://www.sugarcrm.com/crm/product_doc.php
Override Example	`$sugar_config['custom_help_url'] = 'http://www.sugarcrm.com/crm/product_doc.php';`

dashlet_display_row_options (Deprecated in future release)

Description	Determines the number of rows displayed in the 6.x dashlet options. This setting is deprecated as of 7.x. To modify the dashlet row configuration in 7.x you will need to navigate to Admin > Dropdown Editor and edit the dashlet_limit_options list.
Type	Array
Range of values	1, 3, 5, 10
Versions	5.2.0 - 6.7.9
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	5
Override Example	`$sugar_config['dashlet_display_row_options'][] = '20';`

dbconfig

Description	Defines all of the connection parameters for the database server.

Type	Array
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['dbconfig'] = array();`

dbconfig.db_host_instance

Description	Defines the host instance for MSSQL connections.
Type	String
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['dbconfig']['db_host_instance'] = 'SQLEXPRESS';`

dbconfig.db_type

Description	Defines the type of database being used with Sugar. It is important to note that db2 and oracle are only applicable to the Ent and Ult editions of Sugar.
Type	String : Database Enigine
Range of values	'mysql', 'mssql', 'db2', and 'oracle'
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['dbconfig']['db_type'] = 'mysql';`

dbconfig.db_user_name

Description	Defines the user to connect to the Sugar database as.
Type	String : Database User
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['dbconfig']['db_user_name'] = 'sql_user';`

default_currency_significant_digits

Description	Changes the number of significant digits in currency by default.
Type	Integer : Number of significant digits
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	2
Override Example	`$sugar_config['default_currency_significant_digits'] = 2;`

default_date_format

Description	Modifies the default date format for all users.
Type	String : Date format
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	m/d/Y
Override Example	`$sugar_config['default_date_format'] = 'm/d/Y';`

default_decimal_seperator

Description	Sets the character used as a decimal separator for numbers.
Type	String : Text character
Range of values	Any character
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	.
Override Example	`$sugar_config['default_decimal_seperator'] = '.';`

default_email_client

Description	Sets the default email client that opens when users send emails.
Type	String : Email client
Range of values	'sugar', 'external'
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	sugar
Override Example	`$sugar_config['default_email_client'] = 'sugar';`

default_language

Description	Sets each user's default language. Possible values include any language offered by Sugar, such as: 'ar_SA', 'bg_BG', 'ca_ES', 'cs_CZ', 'da_DK', 'de_DE', 'el_EL', 'en_UK', 'en_us', 'es_ES', 'es_LA', 'et_EE', 'fi_FI', 'fr_FR', 'he_IL', 'hu_HU', 'it_it', 'ja_JP', 'ko_KR', 'lt_LT', 'lv_LV', 'nb_NO', 'nl_NL', 'pl_PL',

	'pt_BR', 'pt_PT', 'ro_RO', 'ru_RU', 'sk_SK', 'sq_AL', 'sr_RS', 'sv_SE', 'tr_TR', 'uk_UA', 'zh_CN'
Type	String : Language key
Range of values	Any available language
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	en_us
Override Example	`$sugar_config['default_language'] = 'en_us';`

default_number_grouping_seperator

Description	Sets the character used as the 1000s separator for numbers.
Type	String : Text character
Range of values	Any character
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	,
Override Example	`$sugar_config['default_number_grouping_seperator'] = ',';`

default_permissions

Description	Array that defines the ownership and permissions for directories and files created naturally by the application.
Type	Array
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['default_permissions`

	`'] = array();`

default_permissions.dir_mode

Description	Part of the 'default_permissions' array. Used in UNIX-based systems only to define the permissions on newly created directories. The value is stored in decimal notation while UNIX file permissions are octal. For example, an octal value of 1528 equates to the permissions 2770. Instances running on the On-Demand environment will have this setting enforced as 1528.
Type	Integer : Octal Value
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
On-Demand Value	1528
Override Example	`$sugar_config['default_permissions']['dir_mode'] = 1528;`

default_permissions.file_mode

Description	Part of the 'default_permissions' array. Used in UNIX-based systems only to define the permissions on newly created files. The value is stored in decimal notation while UNIX file permissions are octal. For example, an octal value of 432 in equates to the permissions 660. Instances running on the On-Demand environment will have this setting enforced as 432.
Type	Integer : Octal value
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate

On-Demand Value	432
Override Example	`$sugar_config['default_permissions']['file_mode'] = 432;`

default_permissions.group

Description	Used in UNIX-based systems only to define the group membership of any newly created directories and files. This value should be a group that the Apache user is a member of to help ensure proper functionality. Instances running on the On-Demand environment will have this setting enforced as empty.
Type	String : Web group
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
On-Demand Value	empty
Override Example	`$sugar_config['default_permissions']['group'] = 'apache';`

default_permissions.user

Description	Part of the 'default_permissions' array. Used in UNIX-based systems only to define the ownership of any newly created directories and files. This value should be the Apache user. Instances running on the On-Demand environment will have this setting enforced as empty
Type	String : Web user
Range of values	Apache user
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
On-Demand Value	empty

Override Example	`$sugar_config['default_permissions']['user'] = 'apache';`

default_user_is_admin

Description	Allows for determining whether a user is a system administrator by default.
Type	Boolean
Range of values	true, false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['default_user_is_admin'] = true;`

developerMode

Description	Rebuilds various cached files when a page is accessed. Can be set by an admin in Admin > System Settings. Instances running on the On-Demand environment will have this setting enforced as false.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
On-Demand Value	false
Override Example	`$sugar_config['developerMode'] = true;`

disable_count_query

Description	Removes the count totals from listviews. This is commonly used to prevent performing expensive count queries on the database when loading listviews and subpanels. It is important to note that in 7.x, this parameter will only affect modules running in Backward Compatibility mode.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['disable_count_query'] = true;`

disable_export

Description	Prevents exports of data into .csv files. Normally set in the UI via Admin > Locale.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['disable_export'] = true;`

disable_related_calc_fields

Description	When a calculated field in Sugar uses the related function in the Sugar Logic, this will cause the calculated field to be executed when the related module is updated. This can cause a cascading effect through the system to update related calculated fields. When this happens you may receive a 502 Gateway Error. Please note that this is a global setting that will affect all modules. If you have a calculated field in Accounts that sums up all Opportunities for the account, setting this value to true will no longer update the opportunity account sum in Accounts until the account record itself is modified. However, if this setting is left disabled, the sum would update any time a related opportunity or the account is modified.
Type	Boolean
Range of values	true and false
Versions	6.3.0+
Editions	Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['disable_related_calc_fields'] = true;`

disable_uw_upload

Description	Disables the upgrade wizard from being accessible through the Sugar admin interface. Instances running on the On-Demand environment will have this setting enforced as true.
Type	Boolean
Range of values	true and false
Versions	5.2.0.j+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate

Default Value	false
On-Demand Value	true
Override Example	`$sugar_config['disable_uw_upload'] = true;`

disable_vcr

Description	Disables record paging in the detailview (VCR controls). Increases performance by not loading all records from a listview into memory when accessing the record detailview. In 7.x versions, this setting is only applicable to modules running in Backward Compatibility Mode.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['disable_vcr'] = true;`

dump_slow_queries

Description	Logs slow queries to the sugar log file. Instances running on the On-Demand environment will have this setting enforced as false.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false

On-Demand Value	false
Override Example	`$sugar_config['dump_slow_queries'] = true;`

email_address_separator

Description	Sets the character used to separate email addresses.
Type	String : Text character
Range of values	Any character
Versions	6.4.3+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	,
Override Example	`$sugar_config['email_address_separator'] = ',';`

email_default_client

Description	Sets the default email client for all users.
Type	String : String
Range of values	sugar, external
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	sugar
Override Example	`$sugar_config['email_default_client'] = 'sugar';`

email_default_delete_attachments

Description	When deleting an email, this setting will

	mark all related notes as deleted, and attempt to delete files that are related to those notes.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	true
Override Example	`$sugar_config['email_default_delete_attachments'] = false;`

email_default_editor

Description	Allows configuring the default editor type for email. 'plain' sets the editor to only use plain text. 'html' allows the editor to be html enabled.
Type	String : String
Range of values	'plain' and 'html'
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	html
Override Example	`$sugar_config['email_default_editor'] = 'plain';`

enable_inline_reports_edit

Description	Allows a user to edit specific field types (e.g. dropdowns, text fields) in a rows and columns report without having to navigate directly to the record.
Type	Boolean
Range of values	true and false

Versions	6.3.0+
Editions	Professional, Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['enable_inline_reports_edit'] = true;`

external_cache_disabled

Description	Disables all external caching in Sugar. This is normally set to true to determine if there is a conflict with PHP caching. Instances running on the On-Demand environment will have this setting enforced as true.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
On-Demand Value	true
Override Example	`$sugar_config['external_cache_disabled'] = true;`

external_cache_disabled_apc

Description	Disables APC caching from working with Sugar. Recommended setting is false and is normally set to true to determine if there is a conflict with PHP caching. Instances running on the On-Demand environment will have this setting enforced as true.
Type	Boolean
Range of values	true and false

Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
On-Demand Value	true
Override Example	`$sugar_config['external_cache_disabled_apc'] = false;`

external_cache_disabled_memcache

Description	Disables Memcache caching in Sugar. Recommended setting is false. This setting is normally normall only enabled to determine if there is a conflict with PHP caching. Instances running on the On-Demand environment will have this setting enforced as true.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
On-Demand Value	true
Override Example	`$sugar_config['external_cache_disabled_memcache'] = true;`

external_cache_disabled_memcached

Description	Disables Memcached caching from working with Sugar. Recommended setting is false and is normally set to true to determine if there is a conflict with PHP caching. Instances running on the On-Demand environment will have this setting enforced as true.

Type	Boolean
Range of values	true and false
Versions	6.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
On-Demand Value	true
Override Example	`$sugar_config['external_cache_disabled_memcached'] = false;`

external_cache_disabled_mongo

Description	Disables Mongo caching in Sugar. Recommended setting is false. This setting is normally normall only enabled to determine if there is a conflict with PHP caching. Instances running on the On-Demand environment will have this setting enforced as true.
Type	Boolean
Range of values	true and false
Versions	6.0.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
On-Demand Value	true
Override Example	`$sugar_config['external_cache_disabled_mongo'] = true;`

external_cache_disabled_smash

Description	Disables Smash caching in Sugar. Recommended setting is false and is normally set to true to determine if there is a conflict with PHP caching.

Type	Boolean
Range of values	true and false
Versions	5.2.0c+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['external_cache_disabled_smash'] = false;`

external_cache_disabled_wincache

Description	Disables WinCache caching from working with Sugar. Recommended setting is false and is normally set to true to determine if there is a conflict with PHP caching. Instances running on the On-Demand environment will have this setting enforced as true.
Type	Boolean
Range of values	true and false
Versions	6.0.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
On-Demand Value	true
Override Example	`$sugar_config['external_cache_disabled_wincache'] = false;`

external_cache_disabled_zend

Description	Disables Zend caching from working with Sugar. Recommended setting is false and is normally set to true to determine if there is a conflict with PHP caching. Instances running on the On-Demand environment will have this setting enforced as true.

Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
On-Demand Value	true
Override Example	`$sugar_config['external_cache_disabled_zend'] = false;`

forms

Description	An array defining form requirements.
Type	Array
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['forms'] = array();`

forms.requireFirst

Description	Presents all required fields grouped together in the first panel on the EditView form.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['forms']['requireFirst'] = true;`

hide_admin_backup

Description	Removes the Backups option in the admin menu and also prevents direct access to the feature. Instances running on the On-Demand environment will have this setting enforced as true.
Type	Boolean
Range of values	true and false
Versions	6.5.1+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
On-Demand Value	true
Override Example	`$sugar_config['hide_admin_backup'] = true;`

hide_admin_licensing

Description	Hides the License settings subpanel in the administrative panel. Instances running on the On-Demand environment will have this setting enforced as false.
Type	Boolean
Range of values	true and false
Versions	6.5.0+
Editions	Professional, Corporate, Enterprise, Ultimate
Default Value	false
On-Demand Value	false
Override Example	`$sugar_config['hide_admin_licensing'] = true;`

hide_full_text_engine_config

Description	Determines if the FTS settings are present in the admin search page in Admin > Search. Instances running on the On-Demand environment will have this setting enforced as true.
Type	Boolean
Range of values	true and false
Versions	6.5.15+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
On-Demand Value	true
Override Example	`$sugar_config['hide_full_text_engine_config'] = true;`

hide_subpanels

Description	This setting only applies to modules running in Backward Compatibility Mode. When a DetailView is loaded, all subpanels are collapsed. Collapsing subpanels on load increases performance by not querying for data until a user explicitly expands a subpanel.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['hide_subpanels'] = true;`

hide_subpanels_on_login

Description	This setting only applies to modules running in backward compatibility mode. Collapses subpanels per session. When a DetailView is initially loaded during a session, all subpanels are collapsed. Once explanded, it will remain expanded until the user logs out. Collapsing subpanels on load increases performance by not querying for data until a user explicitly expands a subpanel.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['hide_subpanels_on_login'] = true;`

history_max_viewed

Description	The number of history items from the tracker to display for a user.
Type	Integer
Versions	5.2.0+
Editions	Professional, Corporate, Enterprise, Ultimate
Default Value	50
Override Example	`$GLOBALS['sugar_config']['history_max_viewed'] = 25;`

installer_locked

Description	Sets whether the installer is locked or not. When false, it is possible to access Sugar's initial configuration page to

	reinstall the instance. Instances running on the On-Demand environment will have this setting enforced as true.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	true
On-Demand Value	true
Override Example	`$sugar_config['installer_locked'] = false;`

jobs

Description	Job Queue configurations.
Type	Array
Versions	6.5.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['jobs'] = array();`

jobs.hard_lifetime

Description	Hard deletes all jobs that are older than the hard cutoff. Default is 21 days.
Type	Integer : Days
Versions	6.5.1+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	21
Override Example	`$sugar_config['jobs']['hard_lifetime'] = 21;`

jobs.max_retries

Description	Maximum number of failures for job. Default is 5.
Type	Integer : Number of failures
Versions	6.5.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	5
Override Example	`$sugar_config['jobs']['max_retries'] = 5;`

jobs.min_retry_interval

Description	Minimal interval between job reruns. Default is 30 seconds.
Type	Integer : Seconds
Versions	6.5.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	30
Override Example	`$sugar_config['jobs']['min_retry_interval'] = 30;`

jobs.soft_lifetime

Description	Soft deletes all jobs that are older than cutoff. Default is 21 days.
Type	Integer : Days
Versions	6.5.1+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	7
Override Example	`$sugar_config['jobs']['soft_lifetime'] = 7;`

jobs.timeout

Description	If a job is running longer than the limit, the job is failed by force. Specified in seconds. Default is 3600 seconds (1 hour).
Type	Integer : Seconds
Versions	6.5.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	3600
Override Example	`$sugar_config['jobs']['timeout'] = 86400;`

list_max_entries_per_page

Description	Listview items per page.
Type	String : Records per page
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	20
Override Example	`$sugar_config['list_max_entries_per_page'] = '20';`

list_report_max_per_page

Description	Sets the maximum number of reports that are listed on each page in the Reports module.
Type	Integer : Number of records
Versions	5.2.0+
Editions	Professional, Corporate, Enterprise,

	Ultimate
Default Value	100
Override Example	`$sugar_config['list_report_max_per_page'] = 100;`

lock_homepage (Deprecated in future release)

Description	Prevents users from being able to customize the Homepage layout.
Type	Boolean
Range of values	true and false
Versions	5.2.0 - 6.7.9
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['lock_homepage'] = true;`

logger

Description	An array that defines all of the logging settings.
Type	Array
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['logger'] = array();`

logger.file.dateFormat

Description	The date format for the log file is any value that is acceptable to the PHP strftime() function. The default is '%c'. For a complete list of available date formats, please see the strftime() PHP documentation at

	http://php.net/manual/en/function.strftime.php.
Type	String : Date format
Range of values	Pattern for date format
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	%c
Override Example	`$sugar_config['logger']['file']['dateFormat'] = '%c';`

logger.file.ext

Description	The extension of the log file. The default value is '.log'. Instances running on the On-Demand environment will have this setting enforced as .log.
Type	String : File extension
Range of values	Extension for the log
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	.log
On-Demand Value	.log
Override Example	`$sugar_config['logger']['file']['ext'] = '.log';`

logger.file.maxLogs

Description	When the log file grows to the logger.file.maxSize value, the system will automatically roll the log file. The logger.file.maxLogs value controls the max number of logs that will be saved before it deletes the oldest. The default value is 10.

Type	Integer : Number of logs
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	10
Override Example	`$sugar_config['logger']['file']['maxLogs'] = 10;`

logger.file.maxSize

Description	This value controls the max file size of a log before the system will roll the log file. It must be set in the format '10MB' where 10 is number of MB to store. Always use MB as no other value is currently accepted. To disable log rolling set the value to false. The default value is '10MB'. Instances running on the On-Demand environment will have this setting enforced as 10MB.
Type	String : Size
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	10MB
On-Demand Value	10MB
Override Example	`$sugar_config['logger']['file']['maxSize'] = '10MB';`

logger.file.name

Description	The name of the log file to be written to. Instances running on the On-Demand environment will have this setting enforced as sugarcrm.
Type	String : Filename

Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	sugarcrm
On-Demand Value	sugarcrm
Override Example	`$sugar_config['logger']['file']['name'] = 'sugarcrm';`

logger.file.suffix

Description	The suffix to the file name to track logs chronologically. For instance, if you wanted to append the month and year to a file name, you can change this setting to '%m_%Y'. For a complete list of available date formats, please see the strftime() PHP documentation at http://php.net/manual/en/function.strftime.php. Instances running on the On-Demand environment will have this setting enforced as empty.
Type	String : Suffix pattern
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	empty
On-Demand Value	empty
Override Example	`$sugar_config['logger']['file']['suffix'] = '%m_%Y';`

logger.level

Description	Determines the logging level of the system. The recommended setting is 'fatal'. Instances running on the On-Demand environment will have this setting enforced as fatal.

Type	String : Logging level
Range of values	'debug', 'info', 'warn', 'deprecated', 'error', 'fatal', 'security', 'off'
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	fatal
On-Demand Value	fatal
Override Example	`$sugar_config['logger']['level'] = 'fatal';`

logger_visible

Description	Determines whether the Logger Settings panel is visible to administrators in Admin > System Settings. Instances running on the On-Demand environment will have this setting enforced as false.
Type	Boolean
Range of values	true and false
Versions	6.7.0+
Editions	Professional, Corporate, Enterprise, Ultimate
Default Value	true
On-Demand Value	false
Override Example	`$sugar_config['logger_visible'] = false;`

log_dir

Description	Sets the location in the file system where the Sugar log file will be stored. By default, it is set to '.' meaning that it is stored in the root instance directory. Instances running on the On-Demand

	environment will have this setting enforced as ..
Type	String : Directory Path
Versions	5.2.0+
Editions	Professional, Corporate, Enterprise, Ultimate
Default Value	.
On-Demand Value	.
Override Example	`$sugar_config['log_dir'] = '.';`

log_file

Description	Designates the file name where the instance's logs will be stored. Instances running on the On-Demand environment will have this setting enforced as sugarcrm.log.
Type	String : Name of the log file
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	sugarcrm.log
On-Demand Value	sugarcrm.log
Override Example	`$sugar_config['log_file'] = 'new_sugarcrm.log'`

log_memory_usage

Description	Logs the memory usage. Instances running on the On-Demand environment will have this setting enforced as false.
Type	Boolean
Range of values	true and false
Versions	5.5.0+
Editions	Professional, Corporate, Enterprise,

	Ultimate
Default Value	false
On-Demand Value	false
Override Example	`$sugar_config['log_memory_usage'] = true;`

mark_emails_seen

Description	Determines whether to mark an email as read before importing the email to Sugar during the inbound email import. This is not recommended as an import failure will cause the email to be marked as read which will be skipped during the next inbound email import.
Type	Boolean
Range of values	true and false
Versions	6.5.17+
Editions	Professional, Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['mark_emails_seen'] = true;`

max_dashlets_homepage (Deprecated in future release)

Description	Determines the maximum number of Sugar Dashlets on a users Homepage.
Type	String : Number of dashlets
Versions	5.2.0 - 6.7.9
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	15
Override Example	`$sugar_config['max_dashlets_homepage'] = '20';`

max_session_time

Description	Determines the maximum lock time in seconds between session requests. When a session request is locked for long periods of time, other requests are blocked until it is released. A null value will not implement a max session time. Instances running on the On-Demand environment will have this setting enforced as 1.
Type	Integer : Seconds
Versions	6.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	null
On-Demand Value	1
Override Example	`$sugar_config['max_session_time'] = 1;`

moduleInstaller

Description	Array that defines restrictions on module installations via the Module Loader utility.
Type	Array
Versions	5.2.0.j+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['moduleInstaller'] = array();`

moduleInstaller.disableFileScan

Description	When packageScan is set to 'true', Sugar scans all files in an installable package to ensure that the file extensions are acceptable and that the files do not contain blacklisted class or function calls. Setting the disableFileScan parameter to 'true' avoids this scan from occurring while still enforcing other parameters set.
Type	Boolean
Range of values	true and false
Versions	5.2.0j+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['moduleInstaller']['disableFileScan'] = true;`

moduleInstaller.packageScan

Description	Enables package scanning on any modules uploaded through Module Loader prior to the installation. If the package is found to violate any restrictions of the packageScan, the installation will not proceed and an error report will be generated to the user attempting the install. Instances running on the On-Demand environment will have this setting enforced as true.
Type	Boolean
Range of values	true and false
Versions	5.5.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
On-Demand Value	true
Override Example	`$sugar_config['moduleInstaller']['packageScan'] = true;`

moduleInstaller.validExt

Description	Part of the moduleInstaller array. When moduleInstaller.packageScan is set to true, Sugar will not allow certain file extensions to be present in an installable package. The moduleInstaller.validExt parameter allows you to define additional explicit extensions deemed safe to install on your instance of Sugar. Instances running on the On-Demand environment will have this setting enforced as array('xml').
Type	Array : Extensions
Range of values	File extensions to allow
Versions	6.0.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	array('png', 'gif', 'jpg', 'css', 'js', 'php', 'txt', 'html', 'htm', 'tpl', 'pdf', 'md5', 'xml', 'hbs', 'less', 'wsdl')
On-Demand Value	array('xml')
Override Example	`$sugar_config['moduleInstaller']['validExt'] = array('swf', 'log');`

oracle_enable_ci (Deprecated in future release)

Description	By default, Oracle searching is case sensitive in Sugar, which can be a problem if you are not sure of the case of the data you are searching for. Using this settings, you can turn on insensitive search for Oracle 10g and 11g.
Type	Boolean
Range of values	true and false
Versions	6.1.3 - 7.7.2.0

Editions	Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['oracle_enable_ci'] = true;`

passwordsetting

Description	Defines all of the password requirements for the instance.
Type	Array
Versions	5.5.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['passwordsetting'] = array();`

passwordsetting.forgotpasswordON

Description	Enables the Forgot Password features. When enabled, users will have the ability to reset their own passwords at the Login page. Set in UI via Admin->Password Management.
Type	String
Range of values	0, 1
Versions	5.5.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	1
Override Example	`$sugar_config['passwordsetting']['forgotpasswordON'] = '0';`

passwordsetting.linkexpiration

Description	Determines whether the password reset link expires.
Type	String
Range of values	0, 1
Versions	6.0.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['passwordsetting']['linkexpiration'] = '0';`

passwordsetting.onelower

Description	Configures whether at least one lower-case letter is required in users' passwords.
Type	String
Range of values	0, 1
Versions	6.0.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['passwordsetting']['onespecial'] = '0';`

passwordsetting.onenumber

Description	Configures whether at least one number is required in users' passwords.
Type	String
Range of values	0, 1
Versions	6.0.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	1
Override Example	`$sugar_config['passwordsetting']['onenumber'] = '1';`

passwordsetting.systexpirationtype

Description	Specifies the unit of measurement for passwordsetting.systexpiratiomtime. The available options are: Days (1), Weeks (7) and Months (30). This value can be set in the UI via Admin > Password Management.
Type	Integer
Range of values	1, 7, and 30
Versions	5.5.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	1
Override Example	`$sugar_config['passwordsetting']['systexpirationtype'] = '7';`

quicksearch_querydelay (Deprecated in future release)

Description	The number of seconds that Sugar will wait before searching for records in a quicksearch relate field.
Type	Integer : Seconds
Versions	6.1.3 - 7.0.0
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['quicksearch_querydelay'] = 1;`

require_accounts

Description	Determines whether an account is required for record creation within the system.

Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	true
Override Example	`$sugar_config['require_accounts'] = false;`

SAML_X509Cert

Description	The SAML Certificate Key.
Type	String : SAML Certificate Key
Versions	6.1.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['SAML_X509Cert'] = '-----BEGIN CERTIFICATE-----CERTIFICATE KEY-----END CERTIFICATE-----';`

search_wildcard_infront

Description	In Sugar 7.x+, this setting is only valid for modules running in BWC mode. When enabled, automatically adds a wildcard in front of any searches performed in the application. This setting is not recommended to be enabled as preceding wildcards results in database indices not being utilized and performance decreasing.
Type	Boolean
Range of values	true and false
Versions	6.4.3+
Editions	Community Edition, Professional,

	Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['search_wildcard_infront'] = true;`

session_dir

Description	Directory on the server to store Sugar session data. If left empty, the PHP session settings will be inherited. Instances running on the On-Demand environment will have this setting enforced as empty.
Type	String
Range of values	Directory path
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	empty
On-Demand Value	empty
Override Example	`$sugar_config['session_dir'] = '/tmp/SugarSession/';`

showThemePicker

Description	Removes the theme selection drop down.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	true
Override Example	`$sugar_config['showThemePicker'] = false;`

show_download_tab

Description	Used to determine whether the download tab will appear in the User settings. The download tab provides users with access to Sugar plug-ins and other available downloads.
Type	Boolean
Range of values	true and false
Versions	6.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['show_download_tab'] = true;`

site_url

Description	Current URL of your Sugar instance. This value is critical in its accuracy for multiple points of functionality in the instance.
Type	String : URL
Range of values	Current URL of Sugar
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['site_url'] = 'http://my.sugarinstance.com';`

slow_query_time_msec

Description	Slow query time threshold. Instances running on the On-Demand environment will have this setting enforced as 5000.

Type	String : Milliseconds
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	5000
On-Demand Value	5000
Override Example	`$sugar_config['slow_query_time_msec'] = '1000';`

stack_trace_errors

Description	Displays stack trace of errors.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['stack_trace_errors'] = true;`

studio_max_history

Description	When layout changes are made in Studio, a history of those changes are recorded with each save & deploy under ./custom/history/modules/. The studio_max_history parameter controls how many files Sugar keeps for a particular module. If this parameter is undefined, a default of 50 is observed and to keep all historical Studio actions, set this parameter to 0. Instances running on the On-Demand environment will have this setting enforced as 50.
Type	Integer : Number of files to keep

Range of values	Any integer
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	50
On-Demand Value	50
Override Example	`$sugar_config['studio_max_history'] = 100;`

sugar_version

Description	The current version of Sugar that is being used. This value should not be modified as it is updated in the config when Sugar is upgraded.
Type	String : Version Number
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['sugar_version'] = '6.7.3';`

tmp_dir

Description	The directory path for temporary XML files used by charts and diagnostics.
Type	String : Directory path
Range of values	Filesystem path
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	cache/xml/
Override Example	`$sugar_config['tmp_dir'] = 'cache/xml/';`

tracker_max_display_length

Description	The number of records that will be shown per record in the "Last Viewed" section located under each module tab.
Type	Integer : Number of records
Versions	6.0.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['tracker_max_display_length'] = 45;`

unique_key

Description	Specifies the unique identifier for the instance. This value is used in features such as PHP caching, FTS indexing, and email archiving. It is extremely important that this string is unique from any other instances deployed even if they are only for development purposes only.
Type	String : Unique Identifier
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['unique_key'] = 'c0b5475f3f5b26ddb2976edc8865b5f6';`

upload_badext

Description	An array of extensions that cannot be uploaded in their native file format. Sugar will append a .txt extension to the end of any files with an invalid extension

	to avoid security issues with running unauthorized scripts on an instance.
Type	Array
Range of values	Extensions that cannot be uploaded as is
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['upload_badext'][] = 'swf';`

upload_dir

Description	The directory path where uploaded files are stored for note attachments, documents, and module loadable packages. By default, uploads are stored in the ./upload/ directory.
Type	String
Range of values	Directory path
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	upload/
Override Example	`$sugar_config['upload_dir'] = 'upload/'`

upload_maxsize

Description	The maximum file size that users can upload into Sugar as attachments. When uploading files to Sugar, there are three file size limits to configure. The first two limits are your PHP upload_max_filesize and post_max_size which are configured in your php.ini. The second limit is the sugar configuration setting for upload_maxsize, which will restrict the

	upload limit from within Sugar. This setting can also be modified in the application via Admin > System Settings. The smallest of these three values will be honored when an oversized file has been uploaded.
Type	Integer : Filesize in bytes
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['upload_maxsize'] = 40000000;`

upload_wrapper_class

Description	The name of the class used as upload wrapper.
Type	String : Upload wrapper class
Versions	6.7.0+
Editions	Professional, Corporate, Enterprise, Ultimate
Default Value	UploadStream
Override Example	`$sugar_config['upload_wrapper_class'] = 'SugarUploadS3';`

use_common_ml_dir

Description	A security control that allows you to restrict Module Loader to read modules from a specific directory on the server and disable the ability to upload new modules into the Module Loader. To specify a new directory you will need to populate the config parameter 'common_ml_dir'.
Type	Boolean
Range of values	true and false

Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['use_common_ml_dir'] = true;`

use_php_code_json

Description	Determines if the environment has a valid version of PHP-JSON. This should be determined by the function returnPhpJsonStatus() and shouldn't be overridden.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['use_php_code_json'] = true;`

use_real_names

Description	Display users' full names instead of their User Names in assignment fields.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	true
Override Example	`$sugar_config['use_real_names'] = true;`

use_sprites

Description	A sprite is a two-dimensional image or animation that is integrated into a larger scene. This parameter is used to disable sprites. This is set to true by default (if you have GD libraries installed).
Type	Boolean
Range of values	true and false
Versions	6.4.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['use_sprites'] = false;`

vcal_time

Description	Used to determine the number of months in advance of the current date that the Free/Busy information for calls and meetings will be published. To turn Free/Busy publishing off, set this variable to '0'. The minimum is 1 month; the maximum is 12 months.
Type	String : Months
Range of values	'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', and '12'
Versions	5.2.0.c+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	2
Override Example	`$sugar_config['vcal_time'] = '5';`

verify_client_ip

Description	Whether or not to verify the client IP. Setting this to false will disable the system checking to see if the user is accessing Sugar from the IP address of their last page load.
Type	Boolean
Range of values	true and false
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	true
Override Example	`$sugar_config['verify_client_ip'] = false;`

wl_list_max_entries_per_page

Description	The number of records to be shown per page on the listview of the mobile browser.
Type	Integer : Number of records to display
Versions	5.2.0+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	10
Override Example	`$sugar_config['wl_list_max_entries_per_page'] = 10;`

wl_list_max_entries_per_subpanel

Description	Determines the number of records shown in the subpanels on the DetailView of the mobile browser.
Type	Integer : Records
Versions	5.2.0+

Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	3
Override Example	`$sugar_config['wl_list_max_entries_per_subpanel'] = 3;`

xhprof_config

Description	Configuration settings for xhprof. More information on xhprof can be found at http://pecl.php.net/package/xhprof.
Type	Array
Versions	6.5.10+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['xhprof_config'] = array();`

xhprof_config.enable

Description	Enables the xhprof profiler.
Type	Boolean
Range of values	true and false
Versions	6.5.10+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	false
Override Example	`$sugar_config['xhprof_config']['enable'] = true;`

xhprof_config.flags

Description	The flags for xhprof profiler.
Type	String : xhprof flags

Versions	6.5.10+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['xhprof_config']['flags'] = XHPROF_FLAGS_CPU + XHPROF_FLAGS_MEMORY;`

xhprof_config.ignored_functions

Description	An array of function names to ignore from the profile.
Type	Array : Function names
Versions	6.5.10+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['xhprof_config']['ignored_functions'] = array("function_name");`

xhprof_config.log_to

Description	The path to log the xhprof profiler output to.
Type	String : Directory path
Versions	6.5.10+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Override Example	`$sugar_config['xhprof_config']['log_to'] = '{instance server path}/cache/xhprof';`

xhprof_config.manager

Description	The xhprof manager class to use. Prior to 7.7, specifying values

	xhprof_config.save_to, xhprof_config.mongodb_uri, xhprof_config.mongodb_db, xhprof_config.mongodb_collection, xhprof_config.mongodb_options, and xhprof_config.filter_wt will need to have set xhprof_config.manager set to 'SugarXHprofPerformance'. As of 7.7, setting xhprof_config.manager is not longer required. If you want to customize SugarXHprof, you can create the folder ./custom/include/SugarXHprof/ and create a file with your custom class class name. The custom class will need to extend SugarXHprof. If a custom class doesn't exist or hasn't been specified, SugarXHprof will be used.
Type	String : Class
Versions	6.5.10+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	SugarXHprof
Override Example	`$sugar_config['xhprof_config']['manager'] = 'CustomSugarXHprof';`

xhprof_config.sample_rate

Description	The sample rate of the xhprof profiler. 1/{specified value} requests are profiled. To sample all requests, set this value to 1.
Type	Integer : Sample Rate (1/{value})
Versions	6.5.10+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	10
Override Example	`$sugar_config['xhprof_config']['sample_rate'] = 1;`

xhprof_config.sample_rate

Description	The sample rate of the xhprof profiler. 1/{specified value} requests are profiled. To sample all requests, set this value to 1.
Type	Integer : Sample Rate (1/{value})
Versions	6.5.10+
Editions	Community Edition, Professional, Corporate, Enterprise, Ultimate
Default Value	10
Override Example	`$sugar_config['xhprof_config']['sample_rate'] = 1;`

Last Modified: 12/05/2016 03:01pm

Using Configurator

Overview

The Configuator is a class used to manage the config settings found in 'config.php' and 'config_override.php'.

The Configurator Class

The Configurator class is located in 'modules/Configurator/Configurator.php'. Settings modified using this class are written to 'config_override.php' which is stored in the root directory of your SugarCRM installation.

Retrieving Settings

You can access the SugarCRM config settings by using the global variable 'sugar_config' as shown below:

```
global $sugar_config;
```

```
//Use a specific setting
$MySetting = $sugar_config['MySetting'];
```

If you should need to reload the config settings, this is an example of how to retrieve a specific setting using the configurator:

```
require_once 'modules/Configurator/Configurator.php';

$configuratorObj = new Configurator();

//Load config
$configuratorObj->loadConfig();

//Use a specific setting
$MySetting = $configuratorObj->config['MySetting'];
```

Creating / Updating Settings

To create or update a specific setting, you can specify the new value using the configurator as shown below:

```
require_once 'modules/Configurator/Configurator.php';

$configuratorObj = new Configurator();

//Load config
$configuratorObj->loadConfig();

//Update a specific setting
$configuratorObj->config['MySetting'] = "MySettingsValue";

//Save the new setting
$configuratorObj->saveConfig();
```

Considerations

When looking to store custom settings, the Configurator class will store the settings in the 'config_override.php' file. Alternatively, you can use the Administration class located in ./modules/Administration/Administration.php to store the settings in the 'config' table in the database.

Last Modified: 09/26/2015 04:20pm

DBManagerFactory

Overview

Provides an overview of the DBManagerFactory class that will allow you to generate the appropriate manager for the database you are using.

Instantiating a DB Object

To use the DB object you should use:

```
$GLOBALS['db']
```

If you should need to manually instantiate the DB object, you can use the DBManagerFactory class's getInstance() method. This method will create a reference to the DB object for the instance.

```
$db = DBManagerFactory::getInstance();
```

Querying The Database

Retrieving Results

To query the database looking for a result set, you will use the query() and fetchByAssoc() methods. The query() method will retrieve the results while the fetchByAssoc() method will allow for you to iterate through the results. An Example is below:

```
$sql = "SELECT id FROM accounts WHERE deleted = 0";

$result = $GLOBALS['db']->query($sql);

while($row = $GLOBALS['db']->fetchByAssoc($result) )
{
    //Use $row['id'] to grab the id fields value
    $id = $row['id'];
}
```

Retrieving a Single Result

To retrieve a single result from the database, such as a specific record field, you can use the getOne() method.

```
$sql = "SELECT name FROM accounts WHERE id = '{$id}'";

$name = $GLOBALS['db']->getOne($sql);
```

Limiting Results

To limit the results of a query, you can add a limit to the sql string yourself or use the limitQuery() method. An example is below:

```
$sql = "SELECT id FROM accounts WHERE deleted = 0";
$offset = 0;
$limit = 1;

$result = $GLOBALS['db']->limitQuery($sql, $offset, $limit);

while($row = $GLOBALS['db']->fetchByAssoc($result) )
{
    //Use $row['id'] to grab the id fields value
    $id = $row['id'];
}?
```

Generating SQL Queries

To have Sugar automatically generate SQL queries, you can use some methods from the bean class.

Select Queries

To create a select query you can use the create_new_list_query() method. An example is below:

```
$bean = BeanFactory::newBean($module);

$order_by = '';
$where = '';
$fields = array(
    'id',
    'name',
);
```

```
$sql = $bean->create_new_list_query($order_by, $where, $fields);
```

Count Queries

You can also run the generated SQL through the create_list_count_query() method to generate a count query. An example is below:

```
$bean = BeanFactory::newBean('Accounts');
$sql = "SELECT * FROM accounts WHERE deleted = 0";
$count_sql = $bean->create_list_count_query($sql);
```

Last Modified: 09/26/2015 04:20pm

SugarApplication

Overview

An overview of the SugarApplication class used to help manage tasks for the controller and views.

Displaying Error Messages

Sometimes error messages or notices need to be displayed to users, however, the executing logic cannot be exited. When this happens you can use the method appendErrorMessage() to display messages to the user once the next view loads. An example is shown below:

```
SugarApplication::appendErrorMessage("{$message}");
```

Redirecting Users

To redirect a user to a new page, you can use the redirect() method. This method will handle exiting the code for you. An example is shown below:

```
$urlParameters = array(
```

```
    'module' => $module,
    'action' => $action
);
$url = 'index.php?' . http_build_query($urlParameters);
SugarApplication::redirect($url);
```

Last Modified: 01/08/2017 04:38pm

SugarAutoLoader

This guide provides an overview of the SugarAutoLoader class.

Last Modified: 11/17/2015 11:37pm

Introduction

Overview

The autoloader is an API that allows the unified handling of customizations and customizable metadata, while reducing the number of filesystem accesses and improving performance.

SugarAutoLoader

The SugarAutoLoader class, located in ./include/utils/autoloader.php, keeps a map of files within the Sugar directory that may be loaded . The generated file map is located in the cache directory as ./cache/file_map.php. If this file is manually deleted, it will be automatically rebuilt.

Valid File Extensions

The autoloader will only map files with the following extensions:

- bmp
- css
- gif
- html

- jpg
- js
- less
- override
- php
- png
- tif
- tpl
- xml

*All other file extensions are ignored.

Ignored Directories

The following directories in Sugar are ignored by the autoloader mapping:

- ./.idea/
- ./cache/
- ./custom/backup/
- ./custom/blowfish/
- ./custom/Extension/
- ./custom/history/
- ./custom/modulebuilder/
- ./docs/
- ./examples/
- ./include/HTMLPurifier/
- ./include/nusoap/
- ./include/pclzip/
- ./include/phpmailer/
- ./include/reCaptcha/
- ./include/ytree/
- ./log4php/
- ./portal/
- ./tests/
- ./upload/
- ./Zend/

Initializing the AutoLoader

The init() function will Initialize the loader, register autoload() as an autoloader function, load the bean map, file map, and extension map. This function is called at the beginning of an entryPoint.

```
require_once('include/utils/autoloader.php'); SugarAutoloader::init();
```

Rebuilding the File Map

The autoloader rebuilds the map on module installs, Quick Repair and Rebuilds, and can rebuilt at any time programmatically using the buildCache() function. Same other functions, such as write_array_to_file(), will automatically update the file map. If your code is writing custom files, it is strongly recommended to update the file map immediately. You should also note that if you're hosting your Sugar enviroment On-Demand, you are subject to the Module Loader Restrictions .

The buildCache Function

To programmatically rebuild the file file map, you can use the function as shown below:

```
SugarAutoloader::buildCache();
```

Last Modified: 01/08/2017 04:38pm

Configuration API

Overview

Methods to configure loading paths for the AutoLoader API.

addDirectory($dir)

Adds a directory to the directory map for loading classes. Directories added should include a trailing "/".

```
SugarAutoloader::addDirectory('relative/file/path/');
```

addPrefixDirectory($prefix, $dir)

Adds a prefix and directory to the $prefixMap for loading classes by prefix.

```
SugarAutoloader::addPrefixDirectory('myPrefix',
'relative/file/path/');
```

Last Modified: 09/26/2015 04:20pm

File Check API

Overview

File check methods for use with the AutoLoader API.

existing(...)

Returns an array of filenames that exist in the file map. Accepts any number of arguments of which can be filename or array of filenames. If no files exist, empty array is returned.

```
$files = SugarAutoloader::existing('include/utils.php',
'include/TimeDate.php');
```

existingCustom(...)

This method accepts any number of arguments, each of which can be filename or array of filenames. It will return an array of filenames that exist in the file map, adding also files that exist when custom/ is prepended to them. If the original filename already had custom/ prefix, it is not prepended again. custom/ files are added to the list after the root directory files so that if included in order, they will override the data of the root file. If no files exist, empty array is returned.

```
$files = SugarAutoloader::existingCustom('include/utils.php',
'include/TimeDate.php');
```

existingCustomOne(...)

Returns the last file of the result returned by existingCustom(), or null if none exist. Accepts any number of arguments of which can be filename or array of filenames. Since customized files are placed after the root files, it will return customized file if exists, otherwise root file.

```
$files = SugarAutoloader::existingCustomOne('include/utils.php');
```

You should note that the existingCustomOne() method can be used for loading inline PHP files. An example is shown below:

```
foreach(SugarAutoLoader::existingCustomOne('custom/myFile.php') as
$file)

{

    include $file;

}
```

Alternative to including inline PHP files, loading class files should be done using requireWithCustom() .

fileExists($filename)

Checks if a file exists in the file map. You should note that ".." is not supported by this function and any paths including ".." will return false. The path components should be separated by /. You should also note that multiple slashes are compressed and treated as single slash.

```
$file = 'include/utils.php';

if (SugarAutoloader::fileExists($file))

{

    require_once($file);

}
```

getDirFiles($dir, $get_dirs = false, $extension = null)

Retrieves the list of files existing in the file map under the specified directory. If no files are found, the method will return an empty array. By default, the method will return file paths, however, If $get_dirs is set to true, the method will return only directories. If $extension is set, it would return only files having that specific extension.

```
$files = SugarAutoloader::getDirFiles('include');
```

getFilesCustom($dir, $get_dirs = false, $extension = null)

Retrieves the list of files existing in the file map under the specified directory and under it's custom/ path. If no files are found it will return empty array. By default, the method will return file paths, however, If $get_dirs is set to true, the method will return only directories. If $extension is set, it would return only files having that specific extension.

```
$files = SugarAutoloader::getFilesCustom('include');
```

lookupFile($paths, $file)

Looks up a file in the list of given paths, including with and without custom/ prefix, and return the first match found. The custom/ directory is checked before root files. If no file is found, the method will return false.

```
$paths = array(

  'include',

  'modules',

);



$files = SugarAutoloader::lookupFile($paths, 'utils.php');
```

requireWithCustom($file, $both = false)

If a custom/ override of the file or the file exist, require_once it and return true, otherwise return false. If $both is set to true, both files are required with the root file being first and custom/ file being second. Unlike other functions, this function will actually include the file.

```
$file = SugarAutoloader::requireWithCustom('include/utils.php');
```

You should note that the requireWithCustom() method should be used for loading class files and not inline PHP files. Inline PHP files should be loaded using the existingCustomOne() method.

Last Modified: 09/26/2015 04:20pm

File Map Modification API

Overview

Methods to modify files in the AutoLoader API. All the functions below return true on success and false on failure.

addToMap($filename, $save = true)

Adds an existing file to the file map. If $save is true, the new map will be saved to the disk file map, otherwise it will persist only until the end of the request. This method does not create the file on the filesystem.

```
SugarAutoloader::addToMap('custom/myFile.php');
```

delFromMap($filename, $save = true)

Removes a file from the file map. If $filename points to a directory, this directory and all files under it are removed from the map. If $save is true, the new map will be saved to the disk file map, otherwise it will persist only until the end of the request. This method does not delete the file from the filesystem.

```
SugarAutoloader::delFromMap('custom/myFile.php');
```

ensureDir($dir)

Ensures that a directory exists. If the directory does not exist, it will be created.

```
SugarAutoloader::ensureDir('custom/myDir/');
```

touch($filename, $save = false)

Creates the specified file on the filesystem and adds it to the file map. If $save is true, the new map will be saved to the disk file map, otherwise it will persist only until the end of the request.

```
SugarAutoloader::touch('custom/myFile.php', true);
```

Last Modified: 09/26/2015 04:20pm

Metadata API

Overview

Methods to load metadata for the AutoLoader API.

loadWithMetafiles($module, $varname)

Returns the specified metadata file for a specific module. You should note that due to the scope nature of include(), this function does not load the actual metadata file but will return the file name that should be loaded by the caller.

```
$metadataPath = SugarAutoloader::loadWithMetafiles('Accounts',
'editviewdefs');
```

loadPopupMeta($module, $metadata = null)

Loads popup metadata for either specified $metadata variable or "popupdefs" variable via loadWithMetafiles() and returns it. If no metadata found returns empty array.

```
$popupMetadata = SugarAutoloader::loadPopupMeta('Accounts');
```

loadExtension($extname, $module = "application")

Returns the extension path given the extension name and module. For global extensions the module should be "application" and may be omitted. If the extension has its own module, such as schedulers, it will be used instead of the $module parameter. You should note that due to the scope nature of include(), this function does not load the actual metadata file but return the file name that should be loaded by the caller. If no extension file exists it will return false.

```
//The list of extensions can be found in
./ModuleInstall/extensions.php
$extensionPath = SugarAutoloader::loadExtension('logichooks');
```

Last Modified: 09/26/2015 04:20pm

SugarHttpClient

Overview

The SugarHttpClient class is used to make REST calls.

The SugarHttpClient Class

The SugarHttpClient class is located in 'include/SugarHttpClient.php'. It contains a callRest() method that will allow you to post a request to a REST service via cURL without having to worry about the overhead or the restrictions on the file_get_contents() method when doing outbound webservice calls .

Making a Request

```
<?php

    // specify the REST web service to interact with
    $url = 'http://{sugar_url}/service/v4_1/rest.php';

    // Open a SugarHttpClient session for making the call
    require_once('include/SugarHttpClient.php');

    $client = new SugarHttpClient;

    // Set the POST arguments to pass to the Sugar server
    $parameters = array(
        'user_auth' => array(
            'user_name' => 'username',
            'password' => md5('password'),
        ),
    );

    $json = json_encode($parameters);
    $postArgs = array(
        'method' => 'login',
        'input_type' => 'JSON',
        'response_type' => 'JSON',
        'rest_data' => $json,
    );
```

```
    $postArgs = http_build_query($postArgs);

    // Make the REST call, returning the result
    $response = $client->callRest($url, $postArgs);

    if ( $response === false )
    {
        die("Request failed.\n");
    }

    // Convert the result from JSON format to a PHP array
    $result = json_decode($response);

    if ( !is_object($result) )
    {
        die("Error handling result.\n");
    }

    if ( !isset($result->id) )
    {
        die("Error: {$result->name} - {$result->description}\n.");
    }

    // Get the session id
    $sessionId = $result->id;

?>
```

Last Modified: 01/08/2017 04:38pm

UploadFile

Overview

The UploadFile class handles the various tasks when uploading a file.

Retrieving a Files Upload Location

To retrieve a files upload path, you can use the get_upload_path method and pass in the files GUID id.

```
require_once('include/upload_file.php');
```

```
UploadFile::get_upload_path($file_id);
```

This method will normally return the path as:

```
upload://1d0fd9cc-02e5-f6cd-1426-51a509a63334
```

Retrieving a Files Full File System Location

To retrieve a files full system path path, you can use the get_upload_path and real_path methods as shown below:

```
require_once('include/upload_file.php');

UploadFile::realpath(UploadFile::get_upload_path($file_id));
```

This method will normally return the path as:

```
/Library/WebServer/htdocs/sugarcrm/upload/1d0fd9cc-02e5-f6cd-1426-51a5
09a63334
```

Retrieving a Files Contents

Alternative to using file_get_contents or sugar_file_get_contents, you can retrieve the contents of a file using the get_file_contents method as shown below:

```
require_once('include/upload_file.php');

$file = new UploadFile();

//get the file location
$file->temp_file_location = UploadFile::get_upload_path($file_id);
$file_contents = $file->get_file_contents();
```

Duplicating a File

To duplicate an uploaded file, you can use the duplicate_file method by passing in the files current id and the id you would like it copied to as shown below:

```
require_once('include/upload_file.php');

$uploadFile = new UploadFile();
```

```
$result = $uploadFile->duplicate_file($oldFileId, $newFileId);
```

Last Modified: 09/26/2015 04:20pm

Job Queue

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/17/2015 11:37pm

Introduction

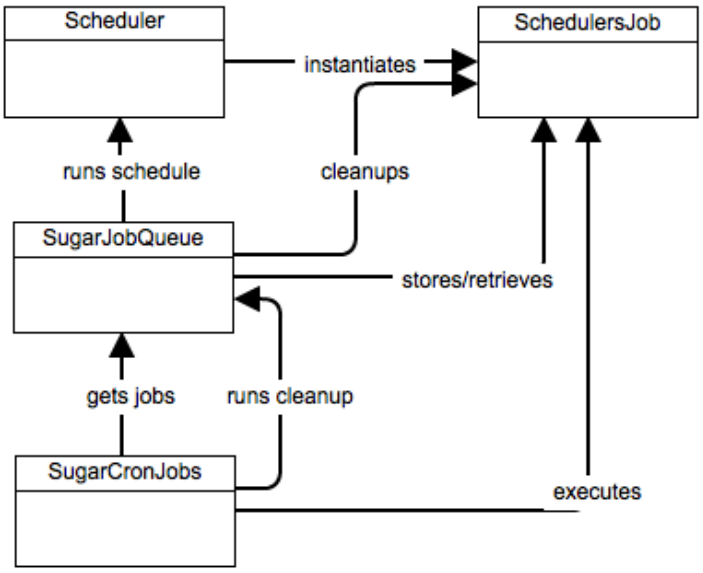
Overview

The Job Queue handles the executing of automated tasks within Sugar.

Components

The Job Queue consists of the followng parts:

- SugarJobQueue - Implements the queue functionality. The queue contains the various jobs.
- SchedulersJob - A single instance of a job. This represents a single executable task and is held in the SugarJobQueue.
- Scheduler - This is a periodically occuring job.
- SugarCronJobs - The cron process that uses SugarJobQueue to run jobs. It is run periodically and does not support parallel execution.



Stages

Schedule Stage

On the scheduling stage (checkPendingJobs in Scheduler class) we check if we have any schedules that are qualified to run at this time and do not have job instance already in the queue. If such schedules exist, each get created a job instance to run immediately.

Execution Stage

The SQL queue table is checked for any jobs in pending status, if such jobs exist their status is set to running, and then the job is executed in accordance to its target and settings.

Cleanup Stage

The queue is checked for jobs that are in running state longer than the defined timeout. Such jobs are failed (they may be requeued if their definition includes requeing on failure).

Last Modified: 08/30/2016 05:25am

Schedulers

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

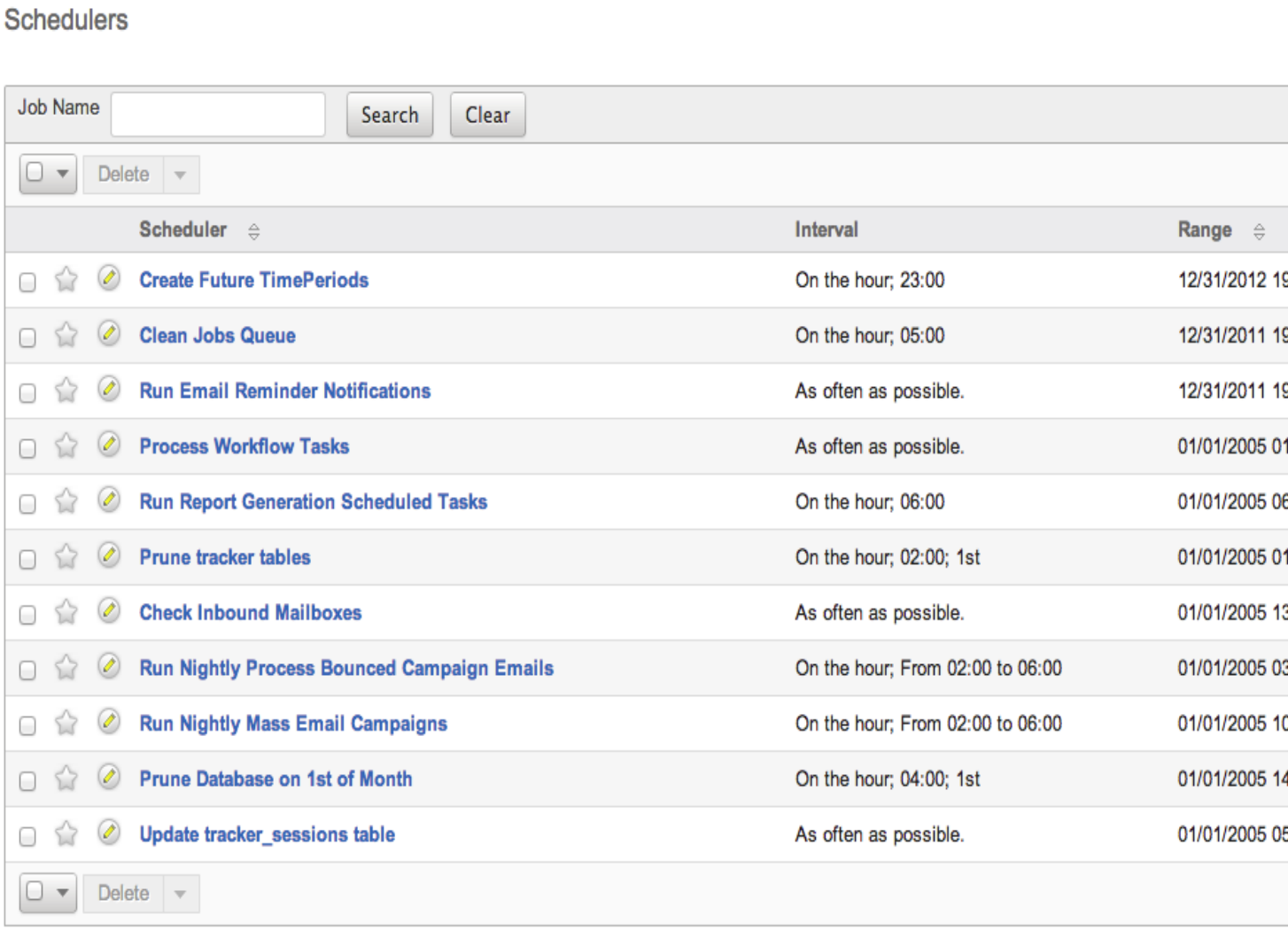
Last Modified: 11/17/2015 11:37pm

Introduction

Scheduler

Sugar provides a Scheduler service that can execute predefined functions asynchronously on a periodic basis. The Scheduler integrates with external UNIX systems and Windows systems to run jobs that are scheduled through those systems. The typical configuration is to have a UNIX cron job or a Windows scheduled job execute the Sugar Scheduler service every couple of minutes. The Scheduler service checks the list of Schedulers defined in the Scheduler Admin screen and executes any that are currently due.

A series of Schedulers are defined by default with every Sugar installation such as Process Workflow Tasks and Run Report Generation Scheduled Tasks.



Config Settings

- cron.max_cron_jobs - Determines the maximum number of jobs executed per cron run.
- cron.max_cron_runtime - Determines the maximum amount of time a job can run before forcing a failure.
- cron.min_cron_interval - Specified the minimum amount of time between cron runs.

Considerations

- When defining schedulers, you should note that the scheduler will be run after the specified time and that this execution may not be exact.

- If you are seeing the message "Job runs too frequently, throttled to protect the system." in your sugar log, the cron is being run too frequently. If you would prefer the cron to run more frequently, you can set the cron.min_cron_interval setting to 0 and disable throttling completely.

Last Modified: 09/26/2015 04:20pm

Custom Schedulers

Overview

How to create a custom scheduler for Sugar 6.3.0+

Scheduler Example

Defining the Job Label

The first step to create a custom scheduler is to create a label extension file. This will add the display text for our scheduler job when creating a new scheduler in Admin > Scheduler. The file path of our file will be in the format of ./custom/Extension/modules/Schedulers/Ext/Language/<language key>.<name>.php. For our example, the file will be nameden_us.custom_job.php.

./custom/Extension/modules/Schedulers/Ext/Language/en_us.custom_job.php

```
<?php

    $mod_strings['LBL_CUSTOM_JOB'] = 'Custom Job';
```

Defining the Job Function

Next, we will define our custom jobs function using the extension framework. The file path of our file will be in the format of ./custom/Extension/modules/Schedulers/Ext/ScheduledTasks/<function_name>.php. For our example, the file will be named custom_job.php. Prior to 6.3.x, job functions were added by creating the file ./custom/modules/Schedulers/_AddJobsHere.php. This method of creating functions is still compatible but is not recommended from a best practices standpoint.

./custom/Extension/modules/Schedulers/Ext/ScheduledTasks/custom_job.php

```
<?php

    array_push($job_strings, 'custom_job');
    function custom_job()
    {
        //logic here
        //return true for completed
        return true;
    }
```

Using the new Job

Once the files are in place, we will need to navigate to Admin > Repair > Quick Repair and Rebuild. This will rebuild the extension directories with our additions. Next, navigate to Admin > Scheduler > Create Scheduler. In the Jobs dropdown, there will be a new custom job in the list.

Last Modified: 09/26/2015 04:20pm

Scheduler Jobs

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 12:41pm

Introduction

Scheduler Jobs

Jobs are the individual runs of the specified function from a scheduler. This article will outline the various parts of a Scheduler Job.

Properties

- name : Name of the job
- scheduler_id : ID of the scheduler that created the job. This may be empty as schedulers are not required to create jobs

- execute_time : Time when job is ready to be executed
- status : Status of the job
- resolution : Notes whether or not the job was successful
- message : Contains messages produced by the job, including errors
- target : Function or URL called by the job
- data : Data attached to the job
- requeue : Boolean to determine whether the job will be replaced in the queue upon failure
- retry_count : Determines how many times the system will retry the job before failure
- failure_count : The number f times the job has failed
- job_delay : The delay (in seconds) between each job run
- assigned_user_id : User ID of which the job will be executed as
- client : The name of the client owning the job
- percent_complete : For postponed jobs, this can be used to determine how much of the job has been completed

Creating a Job

To create job, you must first create an instance of SchedulesJobs class and use submitJob in SugarJobQueue. An example is shown below:

```
<?php
  $jq = new SugarJobQueue();
  $job = new SchedulersJob();
  $job->name = "My Job";
  $job->target = "function::myjob";
  $jobid = $jq->submitJob($job);
  echo "Created job $jobid!\n";
```

Job Targets

Job target contains two components, type and name, separated by "::". Type can be:

- function : Name or static method name (using :: syntax). This function will be passed the job object as the first parameter and if data is not empty, it will be passed as the second parameter.
- url : External URL to call when running the job

Running the Job

The job is run via the runJob() function in SchedulersJob. This function will return a boolean success value according to the value returned by the target function. For URL targets, the HTTP error code being less than 400 will return success.

If the function updated the job status from 'running', the return value of a function is ignored. Currently, the postponing of a URL job is not supported.

Job status

Jobs can be in following statuses:

- queued : The job is waiting in the queue to be executed
- running : The job is currently being executed
- done : The job has executed and completed

Job Resolution

Job resolution is set when the job is finished and can be:

- queued : The job is still not finished
- success : The job has completed successfully
- failure : The job has failed
- partial : The job is partially done but needs to be completed

Last Modified: 09/26/2015 04:20pm

Logic Hooks

Overview

The scheduler jobs module has two additional logic hooks that can be used to monitor jobs.

Hooks

The additional hooks that can be used are.

- job_failure_retry : Executed when a job fails but will be retried
- job_failure : Executed when the fob fails for the final time

You can find more information on these hooks in the Job Queue Logic Hooks section.

Last Modified: 09/26/2015 04:20pm

Examples

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/17/2015 11:37pm

Queuing Logic Hook Actions

Overview

This example will demonstrate how to pass tasks to the job queue. This enables you to send longer running jobs such as sending emails, calling web services, or doing other resource intensive jobs to be handled asynchronously by the cron in the background.

Example

This example will queue an email to be sent out by the cron when an account record is saved.

First, we will create a before_save logic hook on accounts.

./custom/modules/Accounts/logic_hooks.php

```
<?php

    // Do not store anything in this file that is not part of the
array or the hook version. This file will be automatically rebuilt in
```

```
the future.
     $hook_version = 1;
     $hook_array = Array();
     // position, file, function
     $hook_array['before_save'][] = Array();
     $hook_array['before_save'][] = Array(1, 'Queue Job Example',
'custom/modules/Accounts/Accounts_Save.php', 'Accounts_Save',
'QueueJob');
?>
```

In our logic hook, we will create a new SchedulersJob and submit it to the SugarJobQueue targeting our custom AccountAlertJob that we will create next.
./custom/modules/Accounts/Accounts_Save.php

```
<?php
    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
        class Accounts_Save
        {
            function QueueJob(&$bean, $event, $arguments)
        {
            require_once('include/SugarQueue/SugarJobQueue.php');

            // First, let's create the new job
            $job = new SchedulersJob();
            $job->name = "Account Alert Job - {$bean->name}";
            $job->data = $bean->id;
            // key piece, this is data we are passing to the job that
it can use to run it.
            $job->target = "function::AccountAlertJob";
            //function to call global
            $current_user;
            //user the job runs as
            $job->assigned_user_id = $current_user->id;
            // Now push into the queue to run
            $jq = new SugarJobQueue();
            $jobid = $jq->submitJob($job);
        }
    }
?>
```

Next, we will need to define the Job. This will be done by creating a new function to execute our code. We will put this file in the custom/Extension/modules/Schedulers/Ext/ScheduledTasks/ directory with the name AccountAlertJob.php.

./custom/Extension/modules/Schedulers/Ext/ScheduledTasks/AccountAlertJob.php

```
<?php
 function AccountAlertJob($job){
    if (!empty($job->data))
    {
        $bean = BeanFactory::getBean('Accounts', $job->data);
        $emailObj = new Email();
        $defaults = $emailObj->getSystemDefaultEmail();
        $mail = new SugarPHPMailer();
        $mail->setMailerForSystem();
        $mail->From = $defaults['email'];
        $mail->FromName = $defaults['name'];
        $mail->Subject = from_html($bean->name);
        $mail->Body = from_html("Email alert that '{$bean->name}' was
saved");
        $mail->prepForOutbound();
        $mail->AddAddress('example@sugar.crm');
        if($mail->Send())
        {
            //return true for completed
            return true;
        }
    }
    return false;
}
```

Finally, navigate to Admin > Repair > Quick Repair and Rebuild. The system will then generate the file ./custom/modules/Schedulers/Ext/ScheduledTasks/scheduledtasks.ext.php containing our new function. We are now able to queue and run the scheduler job from a logic hook.

Last Modified: 01/08/2017 04:38pm

Language

Overview of working with labels and lists

Last Modified: 09/26/2015 04:20pm

Introduction

Overview

Working with languages.

About

Sugar as an application platform is internationalized and localizable. Data is stored and presented in the UTF8 codepage allowing for all character sets to be used. Sugar provides a language pack framework allowing developers to build support for any language to be used in the display of user interface labels. Every language pack has its own set of display strings which is the basis of language localization. You can add edit or modify these lauguages using this guide.

Last Modified: 01/08/2017 04:38pm

Language Keys

Overview

Sugar as an application platform is internationalized and localizable. Data is stored and presented in the UTF-8 codepage, allowing for all character sets to be used. Sugar provides a language-pack framework that allows developers to build support for any language in the display of user interface labels. Each language pack has its own set of display strings which is the basis of language localization. You can add or modify languages using the information in this guide.

Please scroll to the bottom of this page for additional language topics.

Language Keys

Sugar differentiates languages with unique language keys. These keys prefix the files that correspond with particular languages. For example, the default language for the application is English (US), which is represented by the language key en_us. Any file that contains data specific to the English (US) language begins with the characters en_us. Language label keys that are not recognized will default to the English (US) version.

The following table displays the list of current languages and their corresponding keys:

Language	Language Key
Albanian (Shqip)	sq_AL
Bulgarian (Български)	bg_BG
Catalan (Català)	ca_ES
Chinese (□□□□)	zh_CN
Czech (Česky)	cs_CZ
Danish (Dansk)	da_DK
Dutch (Nederlands)	nl_NL
English (UK)	en_UK
English (US)	en_us
French (Français)	fr_FR
German (Deutsch)	de_DE
Greek (Ελληνικά)	el_EL
Hebrew (□□□□□)	he_IL
Hungarian (Magyar)	hu_HU
Italian (Italiano)	it_it
Japanese (□□□)	ja_JP
Korean (□□□)	ko_KR
Latvian (Latviešu)	lv_LV
Lithuanian (Lietuvių)	lt_LT
Norwegian (Bokmål)	nb_NO
Polish (Polski)	pl_PL
Portuguese (Português)	pt_PT
Portuguese Brazilian (Português Brasileiro)	pt_BR
Romanian (Română)	ro_RO
Russian (Русский)	ru_RU
Serbian (Српски)	sr_RS
Slovak (Slovenčina)	sk_SK
Spanish (Español)	es_ES
Swedish (Svenska)	sv_SE
Turkish (Türkçe)	tr_TR

Change Log

The following table documents historical changes to Sugar's available languages.

Version	Change
6.6.0	Added Albanian language pack.
6.6.0	Added Slovak language pack.
6.6.0	Added Korean language pack.
6.6.0	Added Greek language pack.
6.5.1	Added Latvian language pack.
6.4.0	Added Serbian language pack.
6.4.0	Added English (UK) language pack.
6.4.0	Added Catalan language pack.
6.4.0	Added Portuguese Brazilian language pack.
6.2.0	Added Polish language pack.
6.2.0	Added Hebrew language pack.
6.2.0	Added Czech language pack.
6.1.2	Added Turkish language pack.
6.1.2	Added Swedish language pack.
6.1.2	Added Norwegian language pack.
6.1.2	Added Lithuanian language pack.
6.1.0	Added Chinese language pack.
6.1.0	Added Russian language pack.
6.1.0	Added Romanian language pack.
6.1.0	Added Portuguese language pack.
6.1.0	Added Dutch language pack.
6.1.0	Added Japanese language pack.
6.1.0	Added Italian language pack.
6.1.0	Added Hungarian language pack.
6.1.0	Added French language pack.
6.1.0	Added Spanish language pack.
6.1.0	Added German language pack.
6.1.0	Added Danish language pack.
6.1.0	Added Bulgarian language pack.

Last Modified: 10/22/2016 11:34am

Labels and Lists

Overview

How to work with the application language framework. More information on working wth the module language framework can be found in the Language section under Module Framework.

Language Keys

As Sugar is fully internationalized and localizable, each language is defined by a unique language key. The language keys will prefix any files dealing with languages. An example of a language key is 'en_us' which is used for English (US). For more information regarding please refer to the Language Keys section.

Application Label Framework

$app_list_strings / $app_strings

The $app_list_strings contains the various drop down lists for the system while $app_strings contains the system application labels. The initial set of definitions can be found in the following directory:

./include/language/<language key>.lang.php

As you begin working within the system and deploying modules and lists through Studio, any changes to these labels will be reflected in the corresponding custom directory:

./custom/include/language/<language key>.lang.php

Customizing Labels and Lists

If you are developing a customization and want to be able to create or edit existing label/list values, you will need to work within the extension application directory. To do this you will create a file as follows:

./custom/Extension/application/Ext/Language/<language key>.<unique name>.php

The file will contain your override values. Please note that within this file you will set each label index individually. An example of this is:

```
<?php

    $app_strings['LBL_KEY'] = 'My Display Label';
    $app_list_strings['LIST_NAME']['Key_Value'] = 'My Display Value';
```

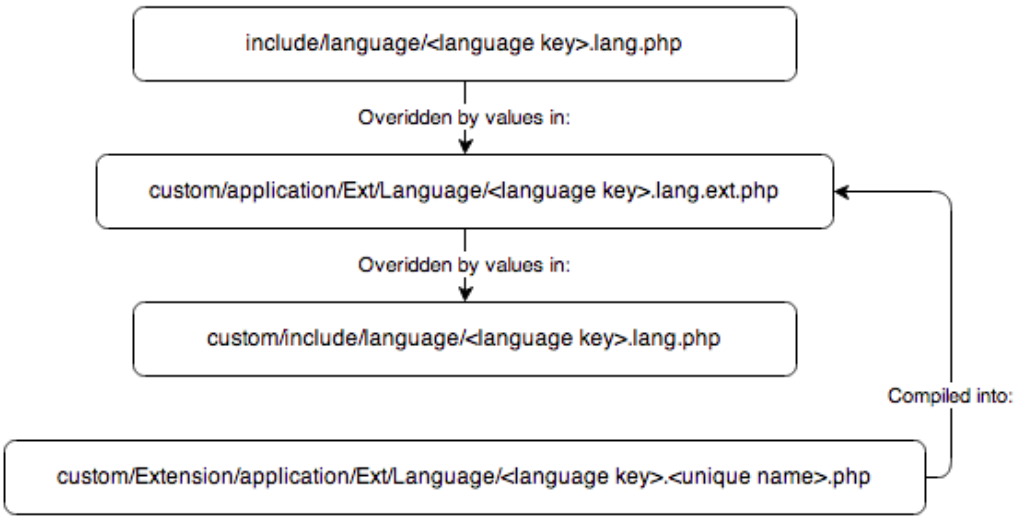
Once the file is created with your adjustments, you will then navigate to Admin > Repair > Quick Rebuild & Repair. This will compile all of the Extension files from:

./custom/Extension/application/Ext/Language/

To:

./custom/application/Ext/Language/<language key>.lang.ext.php

Hierarchy Diagram



Retrieving Labels

There are two ways to retrieve a label. The first is to use the 'translate' function found in 'include/utils.php'. In using this function, the label will be retrieved for the current users language. This function can also be used to retrieve labels from mod_strings, app_strings or app_list_strings.

An example of this is:

```
require_once('include/utils.php');
```

```
$label = translate('LBL_KEY');
```

Alternatively, you can also use the global variable $app_strings as follows:

```
global $app_strings;$label = '';

if (isset($app_strings['LBL_KEY']))
{
    $label = $app_strings['LBL_KEY'];
}
```

Retrieving Lists

There are two ways to retrieve a list. The first is to use the 'translate' function found in 'include/utils.php'. In using this function, the label will be retrieved for the current users language.

An example of this is:

```
require_once('include/utils.php');$list = translate('LIST_NAME');

//You can also retrieve a specific list value this way
$displayValue = translate('LIST_NAME', '', 'Key_Value');
```

Alternatively, you can also use the global variable $app_list_strings as follows:

```
global $app_list_strings;$list = array();

if (isset($app_list_strings['LIST_NAME']))
{
    $list = $app_list_strings['LIST_NAME'];
}
```

Accessing Language Pack Strings with JavaScript

All language pack strings are accessible within the browser-side JavaScript. Use the following JavaScript call to access these strings:

```
// LBL_LOADING string stored in
$app_stringsSUGAR.language.get('app_strings', 'LBL_LOADING');
```

These JavaScript language files are cached. Rebuild the JavaScript files from the Repair console in the Admin section if the language files are changed. This removes the cache files and rebuilds the JavaScript files when needed. It also

increments js_lang_version in sugar_config to recache the JavaScript files in the user's browser.

Last Modified: 09/26/2015 04:20pm

Managing Lists

Overview

Provides a technical overview of the various ways to manage lists.

Modifying Lists

There are 3 ways to modify lists within Sugar.

Studio

Lists can be managed in 2 ways within Studio. If you know the name of the list you would like to edit, you can navigate to the dropdown editor:

Admin > Studio > Dropdown Editor

Alternatively, you can also navigate to your module field using the list and edit it by going to:

Admin > Studio > *module* > Fields > *field*

Direct Modification

There are two ways to directly modify the language strings. The first way is to modify the custom language file.

./custom/include/language/<language key>.lang.php

If you are developing a customization to be distributed and want to be able to create new or override existing list values, you will need to work within the extension application directory. To do this you will create a file as follows:

./custom/Extension/application/Ext/Language/<language key>.<unique name>.php

The file will contain your override values. Please note that within this file you will set each label index individually. An example of this is:

```
<?php
    $app_list_strings['LIST_NAME']['Key_Value'] = 'My Display Value';
```

Once the file is created with your adjustments, you will then navigate to Admin > Repair > Quick Rebuild & Repair. This will compile all of the Extension files from:

./custom/Extension/application/Ext/Language/

To:

./custom/application/Ext/Language/<language key>.lang.ext.php

Dropdown Helper

You can use the Dropdown Helper to manage the lists at a code level. The example below demonstrates how to add and update values for a specific dropdown list.

```
require_once('modules/Studio/DropDowns/DropDownHelper.php');
$dropdownHelper = new DropDownHelper();$parameters = array();
$parameters['dropdown_name'] = 'example_list';$listValues = array(
    'Key_Value_1' => 'Display Value 1',
    'Key_Value_2' => 'Display Value 2',
    'Key_Value_3' => 'Display Value 3');$count = 0;
foreach ($listValues as $key=>$value){
    $parameters['slot_'. $count] = $count;
    $parameters['key_'. $count] = $key;
    $parameters['value_'. $count] = $value;
    //set 'use_push' to true to update/add values while keeping old
values
    $parameters['use_push'] = true;
    $count++;}$dropdownHelper->saveDropDown($parameters);
```

Last Modified: 01/08/2017 04:38pm

Loggers

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting

started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 03:52am

Introduction

Overview

An outline of how the Sugar logger works.

The Sugar Logger

The Sugar Logger allows developers and system administrators to log system events and debugging information into a log file. The Sugar code contains log statements at various points, which are triggered based on the logging level.

For example, to write a message to the sugarcrm.log file when the log level is set to 'fatal', add the following in your code:

```
$GLOBALS['log']->fatal('my fatal message');
```

Logger Level

The logger level determines how much information is written to the sugarcrm.log file. You will find the sugarcrm.log file in the root of your Sugar installation.

Valid values are 'debug', 'info', 'error', 'fatal', 'security', and 'off'. The logger will log information for the specified and higher logging levels. For example if you set the log level to 'error' then you would see logs for 'error', 'fatal', and 'security'. You also may define your own log levels on the fly. For example if you set the value to 'test' then values such as the following would be logged:

```
$GLOBALS['log']->test('This is my test log message');
```

You should avoid using the logging level of 'off'. The default value is 'fatal' and can be defined in the ./config_override.php as follows:

```
$sugar_config['logger']['level'] = 'fatal';
```

You can also force the log level in your code by using:

```
$GLOBALS['log']->setLevel('debug');
```

Log File Name

The default log file name is 'sugarcrm' and can be defined in the ./config_override file as follows:

```
$sugar_config['logger']['file']['name'] = 'sugarcrm';
```

You can also append a suffix to the file name to track logs chronologically. For instance, if you wanted to append the month and year to a file name, the following parameter should be defined in the ./config_override file:

```
$sugar_config['logger']['file']['suffix'] = '%m_%Y';
```

Log File Extension

The default value is '.log'. Therefore the full default log file name is 'sugarcrm.log'. This can be altered with the following parameter in ./config_override.php:

```
$sugar_config['logger']['file']['ext'] = '.log';
```

Log File Date Format

The date format for the log file is any value that is acceptable to the PHP strftime() function. The default is '%c'. For a complete list of available date formats, please see the strftime() PHP documentation. This value can be defined in ./config_override.php as follows:

```
$sugar_config['logger']['file']['dateformat'] = '%c';
```

Max Log File Size

This value controls the max file size of a log before the system will roll the log file. It must be set in the format '10MB' where 10 is number of MB to store. Always use MB as no other value is currently accepted. To disable log rolling set the value to false. The default value is '10MB' and can be defined as follows:

```
$sugar_config['logger']['file']['maxSize'] = '10MB';
```

Max Number of Log Files

When the log file grows to the 'maxSize' value, the system will automatically roll the log file. The 'maxLogs' value controls the max number of logs that will be saved before it deletes the oldest. The default value is 10.

```
$sugar_config['logger']['file']['maxLogs'] = 10;
```

Log Rotation

The Sugar Logger will automatically rotate the logs when the 'maxSize' has been met or exceeded. It will move the current log file to <Log File Name>.1.<Log Extension>. If <Log File Name>.1.<Log Extension> already exists it will rename it to <Log File Name>.2.<Log Extension> prior. This will occur all the way up to the value of 'maxLogs'.

Last Modified: 01/15/2016 10:17pm

Custom Loggers

Overview

How to integrate the logger with alternate logging systems.

Custom Loggers

Custom loggers can be used to write log entries to a centralized application management tool, or to write messages to a developer tool such as FirePHP.

To do this, you can create a new instance class that implements the LoggerTemplate interface. The below code is an example of how to create a FirePHP logger.

You will first need to create your logger class in ./custom/include/SugarLogger/. For this example, we will create ./custom/include/SugarLogger/FirePHPLogger.php.

```
<?php

// change the path below to the path to your FirePHP install
require_once('/path/to/fb.php');
```

```
class FirePHPLogger implements LoggerTemplate{

    /** Constructor */
    public function __construct()
    {
        if (
            isset($GLOBALS['sugar_config']['logger']['default'])
            && $GLOBALS['sugar_config']['logger']['default'] ==
'FirePHP'
            )
        {
            LoggerManager::setLogger('default','FirePHPLogger');
        }
    }

    /** see LoggerTemplate::log() */
    public function log($level, $message)
    {
        // change to a string if there is just one entry
        if ( is_array($message) && count($message) == 1 )
        {
            $message = array_shift($message);
        }

        switch ($level)
        {
            case 'debug':
                FB::log($message);
                break;
            case 'info':
                FB::info($message);
                break;
            case 'deprecated':
            case 'warn':
                FB::warn($message);
                break;
            case 'error':
            case 'fatal':
            case 'security':
                FB::error($message);
                break;
        }
    }
}
```

The only method that needs to be implemented by default is the log() method, which writes the log message to the backend. You can specify which log levels this backend can use in the constuctor by calling the LoggerManager::setLogger() method and specifying the level to use for this logger in the first parameter; passing 'default' makes it the logger for all logging levels.

You will then specify your default logger as 'FirePHP' in your config_override.php file.

```
$sugar_config['logger']['default'] = 'FirePHP';
```

Last Modified: 09/26/2015 04:20pm

Module Builder

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 02:52am

Introduction

Overview

Overview of the Module Builder

Module Builder

The Module Builder functionality allows programmers to create custom modules without writing code, and to create relationships between new and existing CRM modules. To illustrate how to use Module Builder, this article will show how to create and deploy a custom module. In this example, a custom module to track media inquiries will be created to track public relations requests within a CRM system. This use case is an often requested enhancement for CRM systems that applies across industries.

Creating New Modules

Module Builder functionality is managed within the 'Developer Tools' section of Sugar's administration console.

Upon selecting 'Module Builder', the user has the option of creating a "New Package". Packages are a collection of custom modules, objects, and fields that can be published within the application instance or shared across instances of Sugar. Once the user selects "New Package", the user names and describes the type of Custom Module to be created. A package key, usually based on the organization or name of the package creator is required to prevent conflicts between two packages with the same name from different authors. In this case, the package will be named "MediaTracking" to explain its purpose, and a key based on the author name will be used.

Once the new package is created and saved, the user is presented with a screen to create a Custom Module. Upon selecting the "New Module" icon, a screen appears showing six different object templates.

Understanding Object Templates

Five of the six object templates contain pre-built CRM functionality for key CRM use cases. These objects are:"basic", "company", "file", "issue", "person", and "sale". The "basic" template provides fields such as Name, Assigned to, Team, Date Created, and Description. As their title denotes, the rest of these templates contain fields and application logic to describe entities similar to "Accounts", "Documents, "Cases", "Contacts", and "Opportunities", respectively. Thus, to create a Custom Module to track a type of account, you would select the "Company" template. Similarly, to track human interactions, you would select "People".

For the media tracking use case, the user will use the object template "Issue" because inbound media requests have similarities to incoming support cases. In both examples, there is an inquiry, a recipient of the issue, assignment of the issue and resolution. The final object template is named "Basic" which is the default base object type. This allows the administrator to create their own custom fields to define the object.

Upon naming and selecting the Custom Module template named "Issue", the user can further customize the module by changing the fields and layout of the application, and creating relationships between this new module and existing standard or custom modules. This Edit functionality allows user to construct a module that meets the specific data requirements of the Custom Module.

Editing Module Fields

Fields can be edited and created using the field editor. Fields inherited from the custom module's templates can be relabeled while new fields are fully editable. New fields are added using the Add Field button. This displays a tab where you can select the type of field to add as well as any properties that field-type requires.

Editing Module Layouts

The layout editor can be used to change the appearance of the screens within the new module, including the EditView, DetailView and ListView screens. When editing the Edit View or the Detail View, new panels and rows can be dragged from the toolbox on the left side to the layout area on the right. Fields can then be dragged between the layout area and the toolbox. Fields are removed from the layout by dragging them from the layout area to the recycling icon. Fields can be expanded or collapsed to take up one or two columns on the layout using the plus and minus icons. List, Search, Dashlet, and Subpanel views can be edited by dragging fields between hidden/visible/available columns.

Building Relationships

Once the fields and layout of the Custom Module have been defined, the user then defines relationships between this new module and existing CRM data by clicking "View Relationships". The "Add Relationship" button allows the user to associate the new module to an existing or new custom module in the same package. In the case of the Media Tracker, the user can associate the Custom Module with the existing, standard 'Contacts' module that is available in every Sugar installation using a many-to-many relationship. By creating this relationship, end-users will see the Contacts associated with each Media Inquiry. We will also add a relationship to the activities module so that a Media Inquiry can be related to calls, meetings, tasks, and emails.

Publishing and Uploading Packages

After the user has created the appropriate fields, layouts, and relationships for the custom modules, this new CRM functionality can be deployed. Click the "Deploy" button to deploy the package to the current instance. This is the recommended way to test your package while developing. If you wish to make further changes to your package or custom modules, you should make those changes in Module Builder, and click the Deploy button again. Clicking the Publish button generates a zip file with the Custom Module definitions. This is the mechanism for moving the package to a test environment and then ultimately to the production environment.

The Export button will produce a module loadable zip file, similar to the Publish functionality, except that when the zip file is installed, it will load the custom package into Module Builder for further editing. This is a good method for storing the custom package in case you would like to make changes to it in the future on another Sugar instance.

After the new package has been published, the administrator must commit the package to the Sugar system through the Module Loader. The administrator uploads the files and commits the new functionality to the live application instance.

Adding Custom Logic using Code

While the key benefit of the Module Builder is that the Administrator user is able to create entirely new modules without the need to write code, there are still some tasks that require writing PHP code. For instance, adding custom logic or making a call to an external system through a Web Service. This can be done in one of two methods.

Logic Hooks

One way is by writing PHP code that leverages the event handlers, or "logic hooks", available in Sugar. In order to accomplish this, the developer must create the custom code and then add it to the manifest file for the "Media Inquiry" package.

Here is some sample code for a simple example of using the logic hooks. This example adds a time stamp to the description field of the Media Inquiry every time the record is saved.

First, create the file AddTimeStamp.php with the following contents.

```
<?php
//prevents directly accessing this file from a web browser
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
    class AddTimeStamp
    {
        function StampIt(& $focus, $event)
        {
            global $current_user;
            $focus->description .= "Saved on ". date("Y-m-d g:i a"). "
by ". $current_user->user_name;         }
    }
?>
```

Next we will need to register this custom function. There are two ways to handle this.

This first way is to add the logic_hook index to the installdef of the manifest.php file:

```
'logic_hooks' => array(
        array(
            'module' => 'jsche_mediarequest',
            'hook' => 'before_save',
            'order' => 1,
            'description' => 'custom before_save hook',
            'file' =>
'custom/modules/jsche_mediarequest/AddTimeStamp.php',
            'class' => 'AddTimeStamp',
            'function' => 'StampIt',
        ),
  ),
```

Doing this will install the hook when the package is installed. Alternatively, you could create the file logic_hooks.php with the following contents:

```
<?php
// Do not store anything in this file that is not part of the array or
the hook version. This file will be automatically rebuilt in the
future.
     $hook_version = 1;
     $hook_array = Array();

     // position, file, function
     $hook_array['before_save'] = Array();
     $hook_array['before_save'][] = Array(
         1,
          'custom before_save hook',
          'custom/modules/jsche_mediarequest/AddTimeStamp.php',
          'AddTimeStamp',
          'StampIt'
     );
?>
```

Now add these two files to the Media Inquiries zip file you just saved. Create a directory called "SugarModules/custom/" in the zip file and add the two files there. Then modify the manifest.php in the zip file to include the following definition in the $install_defs['copy'] array.

```
array (
    'from' => '<basepath>/SugarModules/custom',
    'to' => 'custom/modules/jsche_mediarequest',
),
```

Custom Bean files

Another method is to add code directly to the custom bean. This is a more complicated approach because it requires understanding the SugarBean class. However it is a far more flexible and powerful approach.

First you must "build" your module. This can be done by either deploying your module or clicking the Publish button. Module Builder will then generate a folder for your package in "custom/modulebuilder/builds/". Inside that folder is where Sugar will have placed the bean files for your new module(s). In this case we want ./custom/modulebuilder/builds/MediaTracking/SugarModules/modules/jsche_mediarequest/

Inside you will find two files of interest. The first one is {module_name}sugar.php. This file is generated by Module Builder and may be erased by further changes in module builder or upgrades to the Sugar application. You should not make any changes to this file. The second is {module_name}.php. This is the file where you make any changes you would like to the logic of your module. To add our timestamp, we would add the following code to jsche_mediarequest.php

```
function save($check_notify = FALSE){
     global $current_user;
     $this->description .= "Saved on " . date("Y-m-d g:i a"). " by ".
$current_user->user_name;
     parent::save($check_notify);
}
```

The call to the parent::save function is critical as this will call on the out of box SugarBean to handle the regular Save functionality. To finish, re-deploy or re-publish your package from Module Builder.

You can now upload this module, extended with custom code logic, into your Sugar application using the Module Loader as described earlier.

Using the New Module

After you upload the new module, the new custom module appears in the Sugar instance. In this example, the new module, named "Media" uses the object template "Issue" to track incoming media inquiries. This new module is associated

with the standard "Contacts" modules to show which journalist has expressed interest. In this example, the journalist has requested a product briefing. On one page, users can see the nature of the inquiry, the journalist who requested the briefing, who the inquiry was assigned to, the status, and the description.

Last Modified: 09/26/2015 04:20pm

Best Practices

Overview

Sugar provides two tools for building and maintaining custom module configurations: Module Builder and Studio. As an administrator of Sugar, it is important to understand the strengths of both tools so that you have a sound development process as you look to build on Sugar's framework.

Goal

This guide will provide you with all the necessary steps from initial definition of your module to the configuration and customization of the module functionality. Follow these tips to ensure your development process will be sound:

Build Your Module in Module Builder

Build the initial framework for your module in Module Builder. Add all the fields you feel will be necessary for the module and construct the layouts with those fields. If possible, it is even better to create your custom modules in a development environment that mirrors your production environment.

Never Redeploy a Package in a Production Environment

Redeploying a package will remove all customizations related to your module in:

- ./modules/
- ./custom/modules/
- ./custom/Extension/modules/

This includes workflows, code customizations, changes through Studio, and much more. It is imperative that this directive is followed to ensure any desired

configurations remain intact. Once the module is deployed, you should use Studio to perform any additional configurations to your module including but not limited to:

- adding a new field
- updating the properties of a field deployed with the module
- changing field layouts
- creating, modifying and removing relationships

Every Module in Module Builder Gets Its Very Own Package

While it is possible to create multiple modules in a package, this can also cause design headaches down the road. If you end up wanting to uninstall a module and it is part of a larger package, all modules in that package would need to be uninstalled. Keeping modules isolated to their own packages allows greater flexibility in the future if a module is no longer needed.

Create Relationships in Studio After the Module Is Deployed

This part is critical for success as relationships created in Module Builder cannot be removed after the module is deployed unless the package is updated and redeployed from Module Builder. Redeploying from Module Builder is what we are trying to avoid as mentioned above. If you deploy the module and then create the relationships in Studio, you can update or remove the relationships via Studio at any future point in time.

Delete the Package from Module Builder Once It Is Deployed

Once the package is deployed, delete it from Module Builder so that it will not accidentally be redeployed. The only exception to this rule is in a development environment as you may want to continue working and testing until you are ready to move the module to your production environent. If you ever want to uninstall the module at a later date, you can do so under Admin > Module Loader.

Last Modified: 11/19/2015 10:28pm

Module Loader

This guide provides an overview of product features and related technologies. In

addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 02:52am

Module Loader Restrictions

Module Loader Restrictions

SugarCRM's hosting objective is to maintain the integrity of the standard Sugar functionality when we upgrade a customer instance, and limit any negative impact our upgrade has on the customer's modifications. All instances hosted on OnDemand have package scanner enabled by default. This setting is not configurable and all packages must pass the package scan for installation to the OnDemand environment.

Access Controls

The Module Loader includes a Module Scanner, which grants system administrators the control they need to determine the precise set of actions that they are willing to offer in their hosting environment. This feature is available in all editions of Sugar. Anyone who is hosting Sugar products can advantage of this feature as well.
The specific Module Loader restrictions for the Sugar Open Cloud are documented in the Sugar Knowledge Base.

Enable Package Scan

Scanning is disabled in default installations of Sugar, and can be enabled through a configuration setting. This setting is added to config.php or config_override.php, and is not available to Administrator users to modify through the Sugar interface. Please note that this setting can only be managed on an On-Site deployment and cannot be disabled for the OnDemand environment.
To enable Package Scan and its associated scans, add this setting to config_override.php:

```
$sugar_config['moduleInstaller']['packageScan'] = true;
```

There are two categories of access controls now available:

- File scanning
- Module Loader actions

Enable File Scan

By enabling Package Scan, File Scan will be performed on all files in the package uploaded through Module Loader. File Scan will be performed when a Sugar administrator attempts to install the package. Please note that these settings can only be managed on an on-site deployment. These settings are not permitted to be modified when hosted on OnDemand.
File Scan performs three checks:

1. File extension must be in the approved list of valid extension types.
    - The default list of valid extension types are detailed below:
        - css
        - gif
        - hbs
        - htm
        - html
        - jpg
        - js
        - md5
        - pdf
        - php
        - png
        - tpl
        - txt
        - xml
2. Files do not contain classes that are considered suspicious, based on a blacklist:
    - Variable classes are not permitted. i.e $class().
    - The default list of blacklisted classes are:
        - lua
        - pclzip
        - reflection
        - reflectionclass
        - reflectionexception
        - reflectionextension
        - reflectionfunction
        - reflectionfunctionabstract
        - reflectionmethod
        - reflectionobject
        - reflectionparameter
        - reflectionproperty
        - reflectionzendextension
        - reflector
        - splfileinfo
        - splfileobject

        - ziparchive
3. Files do not contain function calls that are considered suspicious.
    - Variable functions are not permitted. i.e $func().
    - Backticks (`) are never allowed by File Scan.
    - The default list of blacklisted PHP functions are:
        - addfunction
        - addserver
        - array_diff_uassoc
        - array_diff_ukey
        - array_filter
        - array_intersect_uassoc
        - array_intersect_ukey
        - array_map
        - array_reduce
        - array_udiff
        - array_udiff_assoc
        - array_udiff_uassoc
        - array_uintersect
        - array_uintersect_assoc
        - array_uintersect_uassoc
        - array_walk
        - array_walk_recursive
        - call_user_func
        - call_user_func
        - call_user_func_array
        - call_user_func_array
        - chdir
        - chgrp
        - chmod
        - chroot
        - chwown
        - clearstatcache
        - construct
        - consume
        - consumerhandler
        - copy
        - copy_recursive
        - create_cache_directory
        - create_custom_directory
        - create_function
        - dir
        - disk_free_space
        - disk_total_space
        - diskfreespace
        - eio_busy
        - eio_chmod

- eio_chown
- eio_close
- eio_custom
- eio_dup2
- eio_fallocate
- eio_fchmod
- eio_fchown
- eio_fdatasync
- eio_fstat
- eio_fstatvfs
- eio_fsync
- eio_ftruncate
- eio_futime
- eio_grp
- eio_link
- eio_lstat
- eio_mkdir
- eio_mknod
- eio_nop
- eio_open
- eio_read
- eio_readahead
- eio_readdir
- eio_readlink
- eio_realpath
- eio_rename
- eio_rmdir
- eio_sendfile
- eio_stat
- eio_statvfs
- eio_symlink
- eio_sync
- eio_sync_file_range
- eio_syncfs
- eio_truncate
- eio_unlink
- eio_utime
- eio_write
- error_log
- escapeshellarg
- escapeshellcmd
- eval
- exec
- fclose
- fdf_enum_values
- feof

- fflush
- fgetc
- fgetcsv
- fgets
- fgetss
- file
- file_exists
- file_get_contents
- file_put_contents
- fileatime
- filectime
- filegroup
- fileinode
- filemtime
- fileowner
- fileperms
- filesize
- filetype
- flock
- fnmatch
- fopen
- forward_static_call
- forward_static_call_array
- fpassthru
- fputcsv
- fputs
- fread
- fscanf
- fseek
- fstat
- ftell
- ftruncate
- fwrite
- get
- getbykey
- getdelayed
- getdelayedbykey
- getimagesize
- glob
- header_register_callback
- ibase_set_event_handler
- ini_set
- is_callable
- is_dir
- is_executable
- is_file

- is_link
- is_readable
- is_uploaded_file
- is_writable
- is_writeable
- iterator_apply
- lchgrp
- lchown
- ldap_set_rebind_proc
- libxml_set_external_entity_loader
- link
- linkinfo
- lstat
- mailparse_msg_extract_part
- mailparse_msg_extract_part_file
- mailparse_msg_extract_whole_part_file
- mk_temp_dir
- mkdir
- mkdir_recursive
- move_uploaded_file
- newt_entry_set_filter
- newt_set_suspend_callback
- ob_start
- open
- opendir
- parse_ini_file
- parse_ini_string
- passthru
- passthru
- pathinfo
- pclose
- pcntl_signal
- popen
- preg_replace_callback
- proc_close
- proc_get_status
- proc_nice
- proc_open
- readdir
- readfile
- readline_callback_handler_install
- readline_completion_function
- readlink
- realpath
- realpath_cache_get
- realpath_cache_size

- register_shutdown_function
- register_tick_function
- rename
- rewind
- rmdir
- rmdir_recursive
- session_set_save_handler
- set_error_handler
- set_exception_handler
- set_file_buffer
- set_local_infile_handler
- set_time_limit
- setclientcallback
- setcompletecallback
- setdatacallback
- setexceptioncallback
- setfailcallback
- setserverparams
- setstatuscallback
- setwarningcallback
- setworkloadcallback
- shell_exec
- spl_autoload_register
- sqlite_create_aggregate
- sqlite_create_function
- sqlitecreateaggregate
- sqlitecreatefunction
- stat
- sugar_chgrp
- sugar_chmod
- sugar_chown
- sugar_file_put_contents
- sugar_file_put_contents_atomic
- sugar_fopen
- sugar_mkdir
- sugar_rename
- sugar_touch
- sybase_set_message_handler
- symlink
- system
- tempnam
- timestampnoncehandler
- tmpfile
- tokenhandler
- touch
- uasort

        - uksort
        - umask
        - unlink
        - unzip
        - unzip_file
        - usort
        - write_array_to_file
        - write_encoded_file
        - xml_set_character_data_handler
        - xml_set_default_handler
        - xml_set_element_handler
        - xml_set_end_namespace_decl_handler
        - xml_set_external_entity_ref_handler
        - xml_set_notation_decl_handler
        - xml_set_processing_instruction_handler
        - xml_set_start_namespace_decl_handler
        - xml_set_unparsed_entity_decl_handler
    - The default list of blacklisted class functions are:
        - SugarLogger::setLevel
        - SugarAuotLoader::put
        - SugarAuotLoader::unlink

Modifying File Scan

To disable File Scan, add the following configuration setting to config_override.php:

```
$sugar_config['moduleInstaller']['disableFileScan'] = false;
```

To add more file extensions to the approved list of valid extension types, add the file extensions to the validExt array. The example below adds a .log file extension and .htaccess to the valid extension type list in config_override.php:

```
$sugar_config['moduleInstaller']['validExt'] = array(
    'log',
    'htaccess'
);
```

To add additional function calls to the black list, add the function calls to the blackList array. The example below blocks the strlen() and strtolower() functions from being included in the package:

```
$sugar_config['moduleInstaller']['blackList'] = array(
    'strlen',
    'strtolower'
);
```

To override the black list and allow a specific function to be included in packages, add the function call to the blackListExempt array. The example below removes the restriction for the file_put_contents() function, allowing it to be included in the package:

```
$sugar_config['moduleInstaller']['blackListExempt'] = array(
    'file_put_contents'
);
```

Restricted Copy

To ensure upgrade-safe customizations, it is necessary for system administrators to restrict the copy action to prevent modifying the existing Sugar source code files. New files may be added anywhere (to allow new modules to be added), but any core Sugar source code file must not be overwritten. This is enabled by default when you enable Package Scan.
To disable Restricted Copy, use this configuration setting:
`$sugar_config['moduleInstaller']['disableRestrictedCopy'] = true;`

Module Loader Actions

- pre_execute : Called before a package is installed
- install_copy : Copies files or directories
- install_images : Install images into the custom directory
- install_menus : Installs menus to a specific page or the entire Sugar application
- install_userpage : Adds a section to the User page
- install_dashlets : Installs dashlets into the Sugar application
- install_administration : Installs an administration section into the Admin page
- install_connectors : Installs Sugar Cloud Connectors
- install_vardefs : Modifies existing vardefs
- install_layoutdefs : Modifies existing layouts
- install_layoutfields : Adds custom fields
- install_relationships : Adds relationships
- install_languages : Installs language files
- install_logichooks : Installs a new logic hook
- post_execute : Called after a package is installed

Disabling Module Loader Actions

Certain Module Loader actions may be considered less desirable than others by a System Administrator. A System Administrator may want to allow some Module Loader actions, but disable specific actions that could impact the upgrade-safe integrity of the Sugar instance.
By default, all Module Loader actions are allowed. Enabling Package Scan does not affect the Module Loader actions.
To disable specific Module Loader actions, add the action to the disableActions array. The example below restricts the pre_execute and post_execute actions:

```
$sugar_config['moduleInstaller']['disableActions'] = array(
    'pre_execute',
    'post_execute'
);
```

Disabling Upgrade Wizard

If you are hosting Sugar and wish to lock down the upgrade wizard, you can set disable_uw_upload to true in the config_override. This is intended for hosting providers to prevent unwanted upgrades.

```
$sugar_config['disable_uw_upload'] = true;
```

Last Modified: 01/15/2016 10:09pm

Module Loader Restriction Alternatives

Overview

This article provides workarounds for commonly used functions that are blacklisted by Sugar for the On-Demand environment.

Blacklisted Functions

$variable()

$variable() is sometimes used when trying to dynamically call a function. This is commonly used to retrieve a new bean object.

Restricted use:

```
$module = "Account";
$id = "6468238c-da75-fd9a-406b-50199fe6b5f8";

//creating a new bean
$focus = new $module()

//retrieving a specific record
$focus->retrieve($id);
```

As of 6.3.0, we have implemented newBean and getBean which can be found in the

BeanFactory . Below is the recommended approach:

```
$module = "Accounts";
$id = "6468238c-da75-fd9a-406b-50199fe6b5f8";

//creating a new bean
$focus = BeanFactory::newBean($module);

//or creating a new bean and retrieving a specific record
$focus = BeanFactory::getBean($module, $id);
```

array_filter()

The array_filter filters elements of an array using a callback function. It is restricted from OnDemand use due to its ability to call other restricted functions.

Restricted use:

```
/**
 * Returns whether the input integer is odd
 * @param $var
 * @return int
 */
function odd($var) {
    return($var & 1);
}

$myArray = array("a"=>1, "b"=>2, "c"=>3, "d"=>4, "e"=>5);
$filteredArray = array_filter($myArray, "odd");
```

An alternative to using array_filter is to use a foreach loop.

```
$filteredArray = array();
$myArray = array("a"=>1, "b"=>2, "c"=>3, "d"=>4, "e"=>5);

foreach ($myArray as $key => $value) {
    // check whether the input integer is odd
    if($value & 1) {
        $filteredArray[$key] = $value;
    }
}
```

copy()

The copy method is sometimes used by developers when duplicating files in the uploads directory.

Restricted use:

```
$result = copy($oldFile, $newFile);
```

An alternative to using copy is the duplicate_file method found in the UploadFile class.

```
require_once('include/upload_file.php');

$uploadFile = new UploadFile();
$result = $uploadFile->duplicate_file($oldFileId, $newFileId);
```

file_exists()

file_exists() is commonly used by developers to determine if a file exists in the custom directory.

Restricted use:

```
if(file_exists($file_path)) {
    require_once($file);
}
```

An alternative to using file_exists is the method fileExists which can be found in the SugarAutoLoader .

```
$file = 'include/utils.php';

if (SugarAutoloader::fileExists($file)) {
    require_once($file);
}
```

file_get_contents()

file_get_contents() is used to retrieve the contents of a file.

Restricted use:

```
$file_contents = file_get_contents('file.txt');
```

An alternative to using file_get_contents and sugar_file_get_contents is the method get_file_contents which can be found in the UploadFile class.

```
require_once('include/upload_file.php');

$file = new UploadFile();

//get the file location
$file->temp_file_location = "relative/path/to/file.txt";

//alternatively you can do the following if you know the upload file
id
//$file->temp_file_location = UploadFile::get_upload_path($file_id);
$file_contents = $file->get_file_contents();
```

fwrite()

fwrite() is a function used to write content to a file. As there isn't currently a direct alternative for this function, you may find one of the following a good solution to what you are trying to achieve.

Adding/Removing Logic Hooks

Sometimes a developer will want to append or remove a custom logic hook to:

./custom/modules/<module>/logic_hooks.php

Adding a logic hook can be done by using check_logic_hook_file:

```
//Adding a logic hook
require_once("include/utils.php");

$my_hook = Array(
    999,
    'Example Logic Hook',
    'custom/modules/<module>/my_hook.php',
    'my_hook_class',
    'my_hook_function'
);

check_logic_hook_file("Accounts", "before_save", $my_hook);
```

Removing a logic hook can be done by using remove_logic_hook:

```
//Removing a logic hook
require_once("include/utils.php");

$my_hook = Array(
    999,
    'Example Logic Hook',
    'custom/modules/<module>/my_hook.php',
    'my_hook_class',
    'my_hook_function'
);

remove_logic_hook("Accounts", "before_save", $my_hook);
```

getimagesize()

The getimagesize method is used to retrieve information about an image file.

Restricted use:

```
$img_size = getimagesize($path);
```

If you are looking to verify an image is .png or .jpeg, you can use the verify_uploaded_image method:

```
require_once('include/utils.php');

if (verify_uploaded_image($path)) {
    //logic
}
```

If you are looking to get the mime type of an image, you can use the get_file_mime_type method:

```
$mime_type = get_file_mime_type($path);
```

Last Modified: 11/17/2016 09:26am

Introduction to the Manifest

Overview

The Module Loader relies on a file named manifest.php, which resides in the root

directory of any installable package.

Manifest Definitions

The following section outlines the parameters specified in the $manifest array contained in the manifest file (manifest.php).

$manifest array elements provide the Module Loader with descriptive information about the extension.

name

The name of the package. This is displayed in the module loader.

Example:

```
'name' => 'Example Package',
```

description

A description of the package. This is displayed in the module loader.

Example:

```
'description' => 'Example Package Description',
```

version

The version of the package, i.e. "1.0". This is displayed in the module loader.

Example:

```
'version' => '1.0',
```

author

Contains the author of the package, i.e. "SugarCRM"

Example:

```
'author' => 'SugarCRM',
```

readme

A path (within the package ZIP file) to a readme document that will be displayed during installation.

acceptable_sugar_flavors

Specifies which Sugar Editions the package can be installed on.

Accepted values are:

- CE
- PRO
- CORP
- ENT
- ULT

Example:

```
'acceptable_sugar_flavors' => array(    'CE',    'PRO',    'CORP',
'ENT',    'ULT'),
```

acceptable_sugar_versions

exact_matches

each element in this array should be one exact version string, i.e. "6.4.5" or "6.5.0"

regex_matches

each element in this array should be one regular expression designed to match a group of versions, i.e. "6\.4\.[0-1]$"

Example:

```
'acceptable_sugar_versions' => array(    'exact_matches' => array(
'6.4.0',        '6.4.1',    ),    //or    'regex_matches' => array(
'6\\.4\\.[0-1]$',    ),),
```

copy_files

An array detailing the source and destination of files that should be copied during installation of an upgrade package. Please note that this attribute is only for packages of type 'patch' that are installed through the upgrade wizard. If you are looking to move files into an instance using the module loader you should use the 'copy' attribute in the $installdefs.

from_dir

Used to specify the directory inside the archive from which to copy files from

to_dir

Used to specify the destination directory inside of SugarCRM. Leaving this empty will copy the files and folders located in the from_dir path to SugarCRM's root directory.

force_copy

Used to specify file names that will be forcefully copied over during an upgrade patch. This array is normally empty.

Example:

```
'copy_files' => array (    'from_dir' => '<basepath>/custom/',
'to_dir' => 'custom',     'force_copy' => array (),),
```

dependencies

id_name

The unique name found in $installdefs.

version

The version number.

icon

A path (within the package ZIP file) to an icon image that will be displayed during installation. Examples include: "./patch_directory/icon.gif" and "./patch_directory/images/theme.gif"

is_uninstallable

Setting this directive to TRUE allows the Sugar administrator to uninstall the package. Setting this directive to FALSE disables the uninstall feature.

Example:

```
'is_uninstallable' => true,
```

published_date

The date the package was published.

Example:

```
'published_date' => '2012-07-06 21:01:59',
```

type

The package type. Accepted values are:

langpack

Packages of type langpack will be automatically added to the "Language" dropdown on the Sugar Login screen. They are installed using the Upgrade Wizard.
Example:
```
'type' => 'langpack',
```

module

Packages of type module are installed using the Module Loader.
Example:
```
'type' => 'module',
```

patch

Packages of type patch are installed using the Upgrade Wizard.
Example:
```
'type' => 'patch',
```

theme

Packages of type theme will be automatically added to the "Theme" dropdown on the Sugar Login screen. They are installed using the Upgrade Wizard.
Example:
```
'type' => 'theme',
```

Installdef Definitions

The following section outlines the parameters specified in the $installdef array contained in the manifest file (manifest.php).

$installdef array elements are used by the Module Loader to determine the actual installation steps that need to be taken to install the extension.

id

A unique name for your module.

Example:

```
'id' => 'package_1343150321',
```

connectors

An array detailing files to be copied to the ./custom/modules/Connectors/connectors/ directory as a connector. The path for the connector is determined from the connector 'name' index in the installdefs. With an example name of ext_rest_example, the sources will be installed to ./custom/modules/Connectors/connectors/sources/ext/rest/example/ and the formatters to ./custom/modules/Connectors/connectors/formatters/ext/rest/example/.

name

The class name of the connector. This will also determine the path to install the connector under.

connector

The path to the folder inside of the zip package for the connector files.

formatter

The path to the folder inside of the zip for the formatter files.

```
'connectors' => array(    array(        'name' => 'ext_rest_example',
'connector' => '<basepath>/example/source',        'formatter' =>
'<basepath>/example/formatter',    ),),
```

copy

An array detailing files to be copied to the Sugar directory. A source path and destination path must be specified for each file or directory.
Example:

```
'copy' => array(    0 => array(        'from' =>
'<basepath>/Files/custom/modules/Accounts/accounts_save.php',
'to' => 'custom/modules/Accounts/accounts_save.php',    ),),
```

dashlets

An array detailing files to be copied to the ./custom/modules/Home/Dashlets/ directory. A dashlet name and pack install from path must be specified for each dashlet.

name

The name to install the dashlet under.

from

The directory that contains the dashlet files in the package.

Example:

```
'dashlets' => array(    0 => array(        'name' => 'MyDashlet',
'from' => '<basepath>/MyDashlet',    ),),
```

language

An array detailing individual language files for your module. The source path, destination file, and language pack name must be specified for each language file.

Example:

```
'language'=> array (    array(        'from' =>
'<basepath>/files/Language/Accounts/en_us.lang.php',
'to_module' => 'Accounts',        'language' => 'en_us'    ),),
```

layoutdefs

An array detailing individual layoutdef files, which are used primarily for setting up subpanels in other modules. The source path and destination module must be specified for each layoutdef file.

layoutfields

An array detailing custom fields to be added to existing layouts. The fields will be added to the edit and detail views of target modules.

vardefs

An array detailing individual vardef files, which are used primarily for defining fields and non many-to-many relationships in other modules. The source path and destination module must be specified for each vardef file.

menu

An array detailing the menu file for your new module. A source path and destination module must be specified for each menu file. See the example manifest file below for details.

beans

An array specifying the bean files for your new module. The following sub-directives must be specified for each bean:

module

Your module's name, "Songs"

class
Your module's primary class name, "Song"

path
The path to your bean file where the above class is defined.

tab
Controls whether or not your new module appears as a tab.

relationships

An array detailing relationship files, used to link your new modules to existing modules. A metadata path must be specified for each relationship. See the example manifest file below for details.

custom_fields

An array detailing custom fields to be installed for your new module. The following sub-directives must be specified for each custom field:

name
The internal name of your custom field. Note that your custom field will be referred to as <name>_c, as "_c" indicates a custom field.

label
The visible label of your custom field

type
The type of custom field. Accepted values include text, textarea, double, float, int, date, bool, enum, and relate.

max_size
The custom field's maximum character storage size.

require_option

Used to mark custom fields are either required or option. Accepted values include optional and required.

default_value

Used to specify a default value for your custom field.

ext1

Used to specify a dropdown name (only applicable to enum type custom fields).

ext2

Unused.

ext3

Unused.

audited

Used to denote whether or not the custom field should be audited. Accepted values include 0 and 1.

module

Used to specify the module where the custom field will be added.
Example:

```
'custom_fields' => array (    array (        'name' => 'text_c',
'label' => 'LBL_TEXT_C',        'type' => 'varchar',        'max_size'
=> 255,        'require_option' => 'optional',        'default_value'
=> '',        'ext1' => '',        'ext2' => '',        'ext3' => '',
'audited' => 1,        'module' => 'Accounts',    ),),
```

logic_hooks

An array detailing logic hooks to be installed to the system. The following sub-directives must be specified for each logic hook:

module

Used to specify the module where the logic hook will be added.

hook

Used to specify the logic hook type. Valid types are:

Application Hooks

- after_ui_frame
- after_ui_footer
- server_round_trip

Module Hooks

- after_delete
- after_relationship_add
- after_relationship_delete
- after_restore
- after_retrieve
- after_save
- before_delete
- before_restore
- before_save
- process_record

Job Queue Hooks

- job_failure
- job_failure_retry

User Hooks

- after_load_user
- after_login
- after_logout
- before_logout
- login_failed

order

Used to specify the logic hook execution order.

description

Used to describe the logic hook.

file

Used to specify the file containing the logic hook.

class

Used to specify the class of the logic hook function you would like to execute.

function

Used to specify the logic hook function you would like to execute.

Example:

```
'logic_hooks' => array(    array(        'module' => 'Accounts',
'hook' => 'before_save',        'order' => 99,        'description' =>
'Set Account Name',        'file' =>
'custom/modules/Accounts/Accounts_Save.php',        'class' =>
'Accounts_Save',        'function' => 'SetAccountName',    ),),
```

pre_execute

Used to execute logic from a file (or set of files) before a package is installed. Anything printed to the screen in the script will be displayed when clicking on the 'Display Log' link.

Example:

In manifest.php

```
'pre_execute' => array(    0 =>  '<basepath>/pre_execute.php',),
```

In <installer package>/pre_execute.php

```
<?php    //pre_execute logic    echo 'pre_execute script<br>';    ?>
```

post_execute

Used to execute logic from a file (or set of files) after a package is installed. Anything printed to the screen in the script will be displayed when clicking on the 'Display Log' link.

Example:

```
'post_execute' => array(    0 => '<basepath>/post_execute.php',),
```

In <installer package>/post_execute.php

```
<?php    //post_execute logic    echo 'post_execute script<br>';    ?>
```

pre_uninstall

Used to execute logic from a file (or set of files) before a package is uninstalled. Anything printed to the screen in the script will be displayed when clicking on the 'Display Log' link.

Example:

```
'pre_uninstall' => array(    0 => '<basepath>/pre_uninstall.php',),
```

In <installer package>/pre_uninstall.php

```
<?php    //pre_uninstall logic    echo 'pre_uninstall script<br>';
?>
```

post_uninstall

Used to execute logic from a file (or set of files) after a package is uninstalled. Anything printed to the screen in the script will be displayed when clicking on the 'Display Log' link.

Example:

```
'post_uninstall' => array(    0 => '<basepath>/post_uninstall.php',),
```

In <installer package>/post_uninstall.php

```
<?php    //post_uninstall logic    echo 'post_uninstall script<br>';
?>
```

Last Modified: 01/15/2016 09:56pm

Package Examples

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 02:52am

Creating an Installable Package for a Logic Hook

Overview

This is an overview of how to create a module loadable package that will install a logic hook.

This example will install a sample logic hook to the accounts module that will default the account name to 'My New Account Name ({time stamp})'. The full package is downloadable here for your reference.

For more details on the $manifest or $installdef options, you can visit the 'Intorduction to the Manifest.

Manifest Example

```
<?php

    $manifest = array(
        'acceptable_sugar_flavors' =>
array('CE','PRO','CORP','ENT','ULT'),
        'acceptable_sugar_versions' => array(
            'exact_matches' => array(),
            'regex_matches' => array('6\\.[0-9]\\.[0-9]$'),
        ),
```

```
        'author' => 'SugarCRM',
        'description' => 'Installs a sample logic hook',
        'icon' => '',
        'is_uninstallable' => true,
        'name' => 'Example Logic Hook Installer',
        'published_date' => '2012-07-06 2012 20:45:04',
        'type' => 'module',
        'version' => '1341607504',
    );

    $installdefs = array(
        'id' => 'package_1341607504',
        'copy' => array(
            0 => array(
                'from' =>
'<basepath>/Files/custom/modules/Accounts/accounts_save.php',
                'to' => 'custom/modules/Accounts/accounts_save.php',
            ),
        ),
        'logic_hooks' => array(
            array(
                'module' => 'Accounts',
                'hook' => 'before_save',
                'order' => 99,
                'description' => 'Example Logic Hook - Updates account
name',
                'file' => 'custom/modules/Accounts/accounts_save.php',
                'class' => 'Accounts_Save',
                'function' => 'updateAccountName',
            ),
        ),
    );

?>
```

Logic Hook Example

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class Accounts_Save
    {
        function updateAccountName($bean, $event, $arguments)
```

```
        {
            $bean->name = "My New Account Name (" . time() . ")";
        }
    }

?>
```

Last Modified: 09/26/2015 04:20pm

Creating an Installable Package That Copies Files

Overview

This is an overview of how to create a module loadable package that will copy files into your instance of Sugar. This is most helpful when your instance of Sugar is hosted in our On-Demand environment or by a third party.

For more details on the $manifest or $installdef options, you can visit the 'Introduction to the Manifest File'.

Manifest Example

```
<?php
    $manifest = array(
        'acceptable_sugar_flavors' =>
array('CE','PRO','CORP','ENT','ULT'),
        'acceptable_sugar_versions' => array(
            'exact_matches' => array(),
            'regex_matches' => array('6\\.[0-9]\\.[0-9]$'),
        ),
        'author' => 'SugarCRM',
        'description' => 'Installs updated layouts to the accounts
module',
        'icon' => '',
        'is_uninstallable' => true,
        'name' => 'Example File Installer',
        'published_date' => '2012-11-01 2012 20:45:04',
        'type' => 'module',
        'version' => '1391608631',
    );
```

```
    $installdefs = array(
        'id' => 'package_1391608631',
        'copy' => array(
            0 => array(
                'from' =>
'<basepath>/Files/custom/modules/Accounts/metadata/detailviewdefs.php'
,
                'to' =>
'custom/modules/Accounts/metadata/detailviewdefs.php',
            ),
            1 => array(
                'from' =>
'<basepath>/Files/custom/modules/Accounts/metadata/editviewdefs.php',
                'to' =>
'custom/modules/Accounts/metadata/editviewdefs.php',
            ),
            2 => array(
                'from' =>
'<basepath>/Files/custom/modules/Accounts/metadata/quickcreatedefs.php
',
                'to' =>
'custom/modules/Accounts/metadata/quickcreatedefs.php',
            ),
        ),
    );
?>
```

Last Modified: 09/26/2015 04:20pm

Creating an Installable Package that Creates New Fields

Overview

This is an overview of how to create a Module Loader package that will install custom fields to a module. This example will install a set of fields to the accounts module. The full package is downloadable here for your reference. For more details on the $manifest or $installdef options, you can visit the Introduction to the Manifest File.

Manifest Example

<basepath>/manifest.php

```
<?php

$manifest = array(
    'acceptable_sugar_flavors' => array('CE', 'PRO', 'CORP', 'ENT',
'ULT'),
    'acceptable_sugar_versions' => array(
        'exact_matches' => array(),
        'regex_matches' => array('6\\.[0-9]\\.[0-9]$'),
    ),
    'author' => 'SugarCRM',
    'description' => 'Installs a sample set of custom fields to the
accounts module',
    'icon' => '',
    'is_uninstallable' => true,
    'name' => 'Example Custom Field Installer',
    'published_date' => '2015-05-11 20:45:04',
    'type' => 'module',
    'version' => '1391607505',
);

$installdefs = array(
    'id' => 'package_1341607504',
    'language' => array(
        array(
            'from' =>
'<basepath>/Files/Language/Accounts/en_us.lang.php',
            'to_module' => 'Accounts',
            'language' => 'en_us'
        ),
    ),
    'custom_fields' => array(
        //Text
        array(
            'name' => 'text_field_example_c',
            'label' => 'LBL_TEXT_FIELD_EXAMPLE',
            'type' => 'varchar',
            'module' => 'Accounts',
            'help' => 'Text Field Help Text',
            'comment' => 'Text Field Comment Text',
            'default_value' => '',
            'max_size' => 255,
            'required' => false, // true or false
```

```
            'reportable' => true, // true or false
            'audited' => false, // true or false
            'importable' => 'true', // 'true', 'false', 'required'
            'duplicate_merge' => false, // true or false
        ),
        //DropDown
        array(
            'name' => 'dropdown_field_example_c',
            'label' => 'LBL_DROPDOWN_FIELD_EXAMPLE',
            'type' => 'enum',
            'module' => 'Accounts',
            'help' => 'Enum Field Help Text',
            'comment' => 'Enum Field Comment Text',
            'ext1' => 'account_type_dom', //maps to options - specify
list name
            'default_value' => 'Analyst', //key of entry in specified
list
            'mass_update' => false, // true or false
            'required' => false, // true or false
            'reportable' => true, // true or false
            'audited' => false, // true or false
            'importable' => 'true', // 'true', 'false' or 'required'
            'duplicate_merge' => false, // true or false
        ),
        //MultiSelect
        array(
            'name' => 'multiselect_field_example_c',
            'label' => 'LBL_MULTISELECT_FIELD_EXAMPLE',
            'type' => 'multienum',
            'module' => 'Accounts',
            'help' => 'Multi-Enum Field Help Text',
            'comment' => 'Multi-Enum Field Comment Text',
            'ext1' => 'account_type_dom', //maps to options - specify
list name
            'default_value' => 'Analyst', //key of entry in specified
list
            'mass_update' => false, // true or false
            'required' => false, // true or false
            'reportable' => true, // true or false
            'audited' => false, // true or false
            'importable' => 'true', // 'true', 'false' or 'required'
            'duplicate_merge' => false, // true or false
        ),
        //Checkbox
        array(
            'name' => 'checkbox_field_example_c',
```

```
        'label' => 'LBL_CHECKBOX_FIELD_EXAMPLE',
        'type' => 'bool',
        'module' => 'Accounts',
        'default_value' => true, // true or false
        'help' => 'Bool Field Help Text',
        'comment' => 'Bool Field Comment',
        'audited' => false, // true or false
        'mass_update' => false, // true or false
        'duplicate_merge' => false, // true or false
        'reportable' => true, // true or false
        'importable' => 'true', // 'true', 'false' or 'required'
    ),
    //Date
    array(
        'name' => 'date_field_example_c',
        'label' => 'LBL_DATE_FIELD_EXAMPLE',
        'type' => 'date',
        'module' => 'Accounts',
        'default_value' => '',
        'help' => 'Date Field Help Text',
        'comment' => 'Date Field Comment',
        'mass_update' => false, // true or false
        'required' => false, // true or false
        'reportable' => true, // true or false
        'audited' => false, // true or false
        'duplicate_merge' => false, // true or false
        'importable' => 'true', // 'true', 'false' or 'required'
    ),
    //DateTime
    array(
        'name' => 'datetime_field_example_c',
        'label' => 'LBL_DATETIME_FIELD_EXAMPLE',
        'type' => 'datetime',
        'module' => 'Accounts',
        'default_value' => '',
        'help' => 'DateTime Field Help Text',
        'comment' => 'DateTime Field Comment',
        'mass_update' => false, // true or false
        'enable_range_search' => false, // true or false
        'required' => false, // true or false
        'reportable' => true, // true or false
        'audited' => false, // true or false
        'duplicate_merge' => false, // true or false
        'importable' => 'true', // 'true', 'false' or 'required'
    ),
    //Encrypt
```

```
        array(
            'name' => 'encrypt_field_example_c',
            'label' => 'LBL_ENCRYPT_FIELD_EXAMPLE',
            'type' => 'encrypt',
            'module' => 'Accounts',
            'default_value' => '',
            'help' => 'Encrypt Field Help Text',
            'comment' => 'Encrypt Field Comment',
            'reportable' => true, // true or false
            'audited' => false, // true or false
            'duplicate_merge' => false, // true or false
            'importable' => 'true', // 'true', 'false' or 'required'
        ),
    ),
);

?>
```

Language Example

<basepath>/Files/Language/Accounts/en_us.lang.php

```
<?php

$mod_strings['LBL_TEXT_FIELD_EXAMPLE'] = 'Text Field Example';
$mod_strings['LBL_DROPDOWN_FIELD_EXAMPLE'] = 'DropDown Field Example';
$mod_strings['LBL_CHECKBOX_FIELD_EXAMPLE'] = 'Checkbox Field Example';
$mod_strings['LBL_MULTISELECT_FIELD_EXAMPLE'] = 'Multi-Select Field
Example';
$mod_strings['LBL_DATE_FIELD_EXAMPLE'] = 'Date Field Example';
$mod_strings['LBL_DATETIME_FIELD_EXAMPLE'] = 'DateTime Field Example';
$mod_strings['LBL_ENCRYPT_FIELD_EXAMPLE'] = 'Encrypt Field Example';
```

Last Modified: 09/26/2015 04:20pm

Performance

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 02:52am

PHP Profiling

Overview

Profile the performance of customizations and the overall application.

XHProf

As of the 6.6.2 release, Sugar introduced a new ability to profile via XHProf, which is an easy to use, hierarchical profiler for PHP. This allows developers to better manage and understand customer performance issues introduced by their customizations. This tool enables quick and accurate identification of the sources of performance sinks within the code by generating profiling logs. Profiling gives you the ability to see the call stack for the entire page load with timing details around function and method calls as well as statistics on call frequency.

Assuming XHProf is installed and enabled in your PHP configuration (which you can learn how to do in the PHP Manual), you can enable profiling in Sugar by adding the following parameters to the ./config_override.php file:

```
$sugar_config['xhprof_config']['enable'] = true;

$sugar_config['xhprof_config']['log_to'] = '{instance server
path}/cache/xhprof';

$sugar_config['xhprof_config']['sample_rate'] = 1; // x where x is a
number and 1/x requests are profiled. So to sample all requests set it
to 1

$sugar_config['xhprof_config']['ignored_functions'] = array(); //
array of function names to ignore from the profile (pass into
xhprof_enable)

$sugar_config['xhprof_config']['flags'] = XHPROF_FLAGS_CPU +
XHPROF_FLAGS_MEMORY; // flags for xhprof
```

Please note that with the above 'log_to' parameter, you would need to create the ./cache/xhprof/ directory in your instance directory with proper permissions and ownership for the Apache user. You can also opt to leave the 'log_to' parameter empty and set the logging path via the xhprof.output_dir parameter in the php.ini

file.

Once the above parameters are set, XHProf profiling will log to the indicated directory and allow you to research any performance related issues encountered in the process of developing and maintaining the application.

Last Modified: 01/08/2017 04:38pm

Quicksearch

Overview

Sugar uses a type-ahead combo box system called QuickSearch that utilizes components from the YUI framework.

Quicksearch

The Sugar QuickSearch (SQS) code resides in the file ./include/javascript/quicksearch.js. This file, along with the YUI dependencies are grouped into ./include/javascript/sugar_grp1_yui.js which is loaded in all the main pages of Sugar. A custom YUI AutoComplete widget is used to pull data through an AJAX call to http://{site_url}/index.php. This then accesses the file ./modules/Home/quicksearchQuery.php.

The browser initiates an AJAX call through JavaScript to the server a short delay after the last user input takes place in the browser. A call is then made requesting up to 30 results per result set.

The first ten results are displayed in the browser. If the user refines the search, and the result set is a subset of the first call then no additional call is made. If the result set of the first call is equal to the limit (30), then an additional call is made.

Requirements for a QuickSearch Field

Requirements for a QuickSearch field are listed below:

- Class of the field is set to sqsEnabled
- Field is not set to disabled or readOnly
- JS array sqs_objects is defined and contains the field name
- sugar_grp1_yui.jsmust be loaded on the page

Disabling Automatic Filling

Add the string sqsNoAutofill to the field class to disable automatic filling of the field on blur.

Metadata example:

```
array (
    'name' => 'account_name',
    'displayParams' => array (
        'hiddeButtons' =>'true',
        'size' => 30,
        'class' => 'sqsEnabled sqsNoAutofill'
    )
),
```

Imposing a Delay in QuickSearch

On high performance systems, QuickSearch querying can lead to unintended results because the query returns quickly before the user has finished typing the full string they wish to search. A system administrator can enforce a strict delay on quicksearch fields so that a search will not be performed until X number of seconds have elapsed after a user last stopped typing. To add this parameter, open the ./config_override.php file and enter the following code:

```
$sugar_config['quicksearch_querydelay'] = 1;
```

The value is in whole seconds and it is recommended not to exceed 1 second unless explicitly requested by your users. Anything longer than 1 second could be seen as an annoyance due to an extended wait. Once this code is entered, save the file. Users will need to log out and clear their browser cache before this change is reflected in their instance of Sugar.

Last Modified: 09/26/2015 04:20pm

Schedulers

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 04:52am

SugarBean

This guide provides an overview of the SugarBean.

Last Modified: 11/19/2015 04:52am

CRUD Handling

Overview

The SugarBean class supports all the model operations to and from the database. Any module that interacts with the database utilizes the SugarBean class, which contains methods to create, read/retrieve, update, and delete records in the database.

Create & Read

Creating and Retrieving Records

The BeanFactory class is used for bean loading. The class should be used any time you are creating or retrieving bean objects. It will automatically handle any classes required for the bean. More information on this can be found in the BeanFactory section.

Obtaining the Id of a Recently Saved Bean

For new records, a GUID is generated and assigned to the record id field. Saving new or existing records returns a single value to the calling routine, which is the id attribute of the saved record. The id has the following format:

```
aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee
```

You can retrieve a newly created records id by doing the following:

```
//Create bean
$bean = BeanFactory::newBean($module);
```

```
//Populate bean fields
$bean->name = 'Example Record';

//Save
$bean->save();

//Retrieve the bean id
$record_id = $bean->id;
```

Saving a Bean with a Specific Id

Saving a record with a specific id requires the id and new_with_id attribute of the bean to be set as follows:

```
//Create bean
$bean = BeanFactory::newBean($module);

//Set the record id
$bean->id = '38c90c70-7788-13a2-668d-513e2b8df5e1';
$bean->new_with_id = true;

//Populate bean fields
$bean->name = 'Example Record';

//Save
$bean->save();
```

Setting new_with_id to true prevents the save method from creating a new id value and uses the assigned id attribute. If the id attribute is empty and the new_with_id attribute is set to true, the save will fail.

Identifying New from Existing Records

To identify whether or not a record is new or existing, you can check the fetched_rows property. If the $bean has a fetched_row, it was loaded from the database. An example is shown below:

```
if (!isset($bean->fetched_row['id']))
{
    //new record
}

else
```

```
{
    //existing record
}
```

Update

Updating a Bean

Updating a bean can be done by fetching a record and then updating the property:

```
//Retrieve bean
$bean = BeanFactory::getBean($module, $id);

//Fields to update
$bean->name = 'Updated Name';

//Save
$bean->save();
```

Updating a Bean without Modifying the Date Modified

The SugarBean class contains an attribute called update_date_modified, which is set to true when the class is instantiated and means that the date_modified attribute is updated to the current database date timestamp. Setting the update_date_modified to false would result in the date_modified attribute not being set with the current database date timestamp.

```
//Retrieve bean
$bean = BeanFactory::getBean($module, $id);

//Set modified flag
$bean->update_date_modified = false;

//Fields to update
$bean->name = 'Updated Name';

//Save
$bean->save();
```

Deleting a Bean

Deleting a bean can be done by fetching a record and then setting the deleted property to 1:

```
//Retrieve bean
$bean = BeanFactory::getBean($module, $id);

//Set deleted to true
$bean->deleted = 1;

//Save
$bean->save();
```

Last Modified: 09/26/2015 04:20pm

Fetching Relationships

Overview

The SugarBean class supports fetching related information from the database.

Fetching Related Records

To fetch related records, you will need to load the relationship using the link.

```
//If relationship is loaded
if ($bean->load_relationship($link))
{
    //Fetch related beans
    $relatedBeans = $bean->$link->getBeans();
}
```

An example of this is to load the contacts related to an account:

```
//Load Account
$bean = BeanFactory::getBean('Accounts', $id);

//If relationship is loaded
if ($bean->load_relationship('contacts'))
{
    //Fetch related beans
    $relatedBeans = $bean->contacts->getBeans();
}
```

Fetching a Parent Record

Fetching a parent record is very similar to fetching child records in that you will still need to load the relationship using the link.

```
//If relationship is loaded
if ($bean->load_relationship($link))
{
    //Fetch related beans
    $relatedBeans = $bean->$link->getBeans();

    $parentBean = false;
    if (!empty($relatedBeans))
    {
        //order the results
        reset($relatedBeans);

        //first record in the list is the parent
        $parentBean = current($relatedBeans);
    }
}
```

An example of this is to load a contact and fetch its parent account:

```
//Load Contact
$bean = BeanFactory::getBean('Contacts', $id);

//If relationship is loaded
if ($bean->load_relationship('accounts'))
{
    //Fetch related beans
    $relatedBeans = $bean->accounts->getBeans();

    $parentBean = false;
    if (!empty($relatedBeans))
    {
        //order the results
        reset($relatedBeans);

        //first record in the list is the parent
        $parentBean = current($relatedBeans);
    }
}
```

Last Modified: 09/26/2015 04:20pm

SugarPDF

Working with TCPDF and the SugarPDF Framework.

Last Modified: 11/19/2015 02:52am

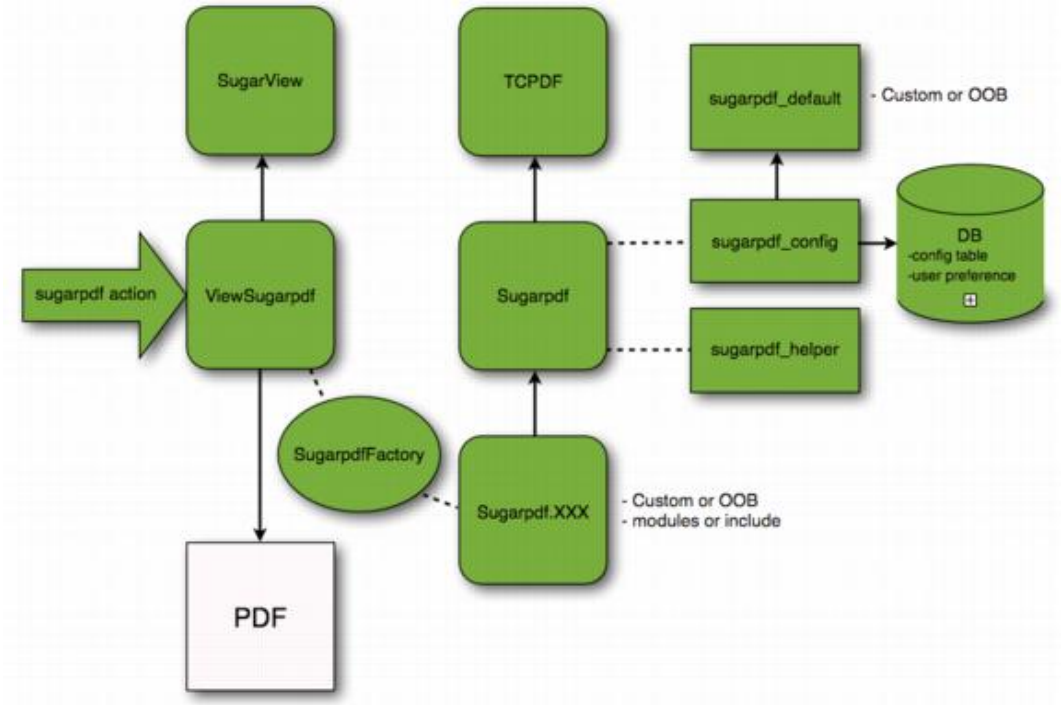
Introduction

Overview

An overview of the PDF framework.

SugarPDF Architecture

The PDF framework leverages the SugarCRM MVC architecture and is built upon the TCPDF library.



TCPDF Class

The TCPDF class, located in ./include/tcpdf/tcpdf.php, is a third party PHP class for generating PDF documents. We use and extend upon this class to generate the PDFs users receive from the system. For more documentation on working with TCPDF, you can visit http://www.tcpdf.org.

SugarPDF Class

The SugarPDF class, located in ./include/Sugarpdf/Sugarpdf.php, extends the TCPDF class. This is the basis on which we generate all PDFs within Sugar. As we have extended TCPDF, we have overridden the following methods:

- Header : Altered to enable the use of the Sugar image directories for generating the PDF header.
- SetFont : Altered to enable the use of the Sugar Font directories.
- Cell : Altered to apply the prepare_string() function to data being printed by the PDF.
- getNumLines : Adjusted for handling record counts.

Methods added to the class for development use are:

- predisplay : Used to preprocessing before the display method is called. It is intended to setup the general PDF document properties like margin, footer, header, etc.
- display : Performs the actual generating of the PDF. This is where the logic to display output to the PDF is placed.
- process : Calls the predisplay and display methods.
- writeCellTable : Method to print a table using the cell print method of TCPDF
- writeHTMLTable : Method to print a table using the writeHTML print method of TCPDF

SugarPDF View

The ViewSugarpdf class, located in ./include/MVC/View/views/view.sugarpdf.php, is a view used to generate the PDF document. As with all SugarViews, the stock ViewSugarpdf class can be overridden by files in the following directories:

- ./custom/modules/<module>/views/view.sugarpdf.php
- ./modules/<module>/views/view.sugarpdf.php

Modules with a SugarPDF view can be accessed using the following URL format:

```
http://{sugar
url}/index.php?module=<module>&action=sugarpdf&sugarpdf=<template>
```

As you can see, the view is accepting 3 query string parameters:

- module=<module> : Determines the module containing the PDF generation classes
- action=sugarpdf : Specifies the SugarPDF action view
- sugarpdf=<template> : Specifies the template name for the PDF we want to generate

More information on creating a SugarPDF View can be found in the Generating PDFs section.

SugarPDF Templates

SugarPDF templates are templates used for generating the PDF documents that users receive. The templates can typically be found in the following directory paths:

- ./include/Sugarpdf/sugarpdf/sugarpdf.<template>.php
- ./modules/<module>/sugarpdf/sugarpdf.<template>.php
- ./custom/modules/<module>/sugarpdf/sugarpdf.<template>.php

These template classes extend the Sugarpdf class and are accessed by name via the "sugarpdf" query string parameter when calling the SugarPDF view:

```
http://{sugar
url}/index.php?module=<module>&action=sugarpdf&sugarpdf=<template>
```

When the SugarPDF action is called, the most relevant SugarPDF template class is selected by the SugarPDF Factory .

SugarPDF Factory

The SugarPDF Factory, located in ./include/Sugarpdf/SugarpdfFactory.php, is used

by the SugarPDF view to find the most relevant template class given a module and template name. If a template is not found, then the default SugarPDF class is used.

The template is searched for in the following directories:

- ./custom/modules/<module>/sugarpdf/sugarpdf.<template>.php
- ./modules/<module>/sugarpdf/sugarpdf.<template>.php
- ./custom/include/Sugarpdf/sugarpdf/sugarpdf.<template>.php
- ./include/Sugarpdf/sugarpdf/sugarpdf.<template>.php
- ./include/Sugarpdf/sugarpdf.php

SugarPDF Helpers

To help with the formatting of data for PDF documents, you can use the helper functions located in ./include/Sugarpdf/SugarpdfHelper.php. This file is included by SugarPDF by default and includes several functions you can use in your templates.

Available functions :

- wrapTag, wrapTD, wrapTable, etc. : These functions help to present your data using HTML.
- prepare_string : This function prepares a string for use when PDF printing.
- format_number_sugarpdf : This function is used when formatting numbers or currencies.

Font Manager

The FontManager, located in ./include/Sugarpdf/FontManager.php, is a stand-alone class that manages all the fonts for TCPDF. It contains the ability to:

- List all the available fonts
- Fetch the details for each font
- Add new fonts to the custom font directory from a font or metric file
- Delete a font from the custom font directory

More information can be found in the Font Manager section.

Last Modified: 01/15/2016 10:01pm

Generating PDFs

Overview

Overview of how to create a PDF document.

Generating a PDF

To generate a PDF in Sugar, you will need to create a template that extends the SugarPDF class. To do this you will need to create a file in the format of:

./custom/modules/<module>/sugarpdf/sugarpdf.<pdf view>.php

```
<?php

if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

require_once('include/Sugarpdf/Sugarpdf.php');

class <module>Sugarpdf<pdf view> extends Sugarpdf
{
    /**
     * Override
     */
    function process(){
        $this->preDisplay();
        $this->display();
        $this->buildFileName();
    }

    /**
     * Custom header
     */
    public function Header()
    {
        $this->SetFont(PDF_FONT_NAME_MAIN, 'B', 16);
        $this->MultiCell(0, 0, 'TCPDF Header',0,'C');
        $this->drawLine();
    }

    /**
```

```
 * Custom header
 */
public function Footer()
{
    $this->SetFont(PDF_FONT_NAME_MAIN, '', 8);
    $this->MultiCell(0,0,'TCPDF Footer', 0, 'C');
}

/**
 * Predisplay content
 */
function preDisplay()
{
    //Adds a predisplay page
    $this->AddPage();
    $this->SetFont(PDF_FONT_NAME_MAIN,'',PDF_FONT_SIZE_MAIN);
    $this->Ln();
    $this->MultiCell(0,0,'Predisplay Content',0,'C');
}

/**
 * Main content
 */
function display()
{
    //add a display page
    $this->AddPage();
    $this->SetFont(PDF_FONT_NAME_MAIN,'',PDF_FONT_SIZE_MAIN);
    $this->Ln();
    $this->MultiCell(0,0,'Display Content',0,'C');
}

/**
 * Build filename
 */
function buildFileName()
{
    $this->fileName = 'example.pdf';
}

/**
 * This method draw an horizontal line with a specific style.
 */
protected function drawLine()
{
    $this->SetLineStyle(array('width' => 0.85 /
```

```
$this->getScaleFactor(), 'cap' => 'butt', 'join' => 'miter', 'dash' =>
0, 'color' => array(220, 220, 220)));
        $this->MultiCell(0, 0, '', 'T', 0, 'C');
    }
}
```

This file example contains the basic markup to generate a PDF. The main things to note are the Header(), Footer() and display() methods as they contain most of the styling and display logic. The method buildFileName() method will generate the document name when downloaded by the user.

Once the file is in place, you can access it by navigating to:

```
http://{sugar
url}/index.php?module=<module>&action=sugarpdf&sugarpdf=<pdf action>
```

This will generate and download the PDF document as 'example.pdf'.

Last Modified: 09/26/2015 04:20pm

PDF Settings

Overview

An overview of how to manage PDF settings.

PDF Setting Framework

This section will outline how to configure the general PDF settings. There are the settings that determine widths, heights, images and pdf metadata as well as the UI configurations found in:

```
Admin > PDF Manager > Edit Report PDF Template
```

Settings

The default PDF settings for TCPDF can be found in ./include/Sugarpdf/sugarpdf_default.php. You can add additional custom settings by creating the following file:

./custom/include/Sugarpdf/sugarpdf_default.php

```
<?php
    $sugarpdf_default["PDF_NEW_SETTING"] = "Value";
```

You should note that values specified here will be the default values. Once edited, the updated values are stored in the database config table under the category "sugarpdf" and a name matching the setting name.

```
category: sugarpdf
name: pdf_new_setting
value: Value
```

Displaying and Editing Settings

A select set of settings can be edited within the Sugar UI by navigating to:

```
Admin > PDF Manager > Edit Report PDF Template
```

The settings here are managed in the file ./modules/Configurator/metadata/SugarpdfSettingsdefs.php. A brief description of the settings parameters are listed below:

- label : This is the display label for the setting.
- info_label : Hover info text for the display label.
- value : The PDF Setting.
- class : Determines which panel the setting resides in. Possible values are 'basic' and 'logo'. 'advanced' is not currently an available value.
- type : Determines the settings display widget. Possible values are: 'image', 'text', 'percent', 'multiselect', 'bool', 'password', and 'file'.

Custom settings can be added to this page by creating ./custom/modules/Configurator/metadata/SugarpdfSettingsdefs.php and specifying a new setting index. An example is below:

./custom/modules/Configurator/metadata/SugarpdfSettingsdefs.php

```
<?php
    //Retrieve setting info from db
    defineFromConfig("PDF_NEW_SETTING",
$sugarpdf_default["PDF_NEW_SETTING"]);
    //Add setting display
    $SugarpdfSettings['sugarpdf_pdf_new_setting'] = array(
        "label" => $mod_strings["LBL_PDF_NEW_SETTING_TITLE"],
```

```
        "info_label" => $mod_strings["LBL_PDF_NEW_SETTING_INFO"],
        "value" => PDF_NEW_SETTING,
        "class" => "basic",
        "type" => "text",
    );
```

You should note that the $SugarpdfSettings index should following the format sugarpdf_pdf_<setting name>. If your setting does not follow this format, it will not be saved or retrieved from the database correctly.

Once the setting is defined, you will need to define the display text for the UI setting.

./custom/modules/Configurator/language/en_us.lang.php

```
<?php
    $mod_strings['LBL_PDF_NEW_SETTING_TITLE'] = 'PDF New Setting';
    $mod_strings['LBL_PDF_NEW_SETTING_INFO'] = 'Display info for the
new PDF setting';
```

Once finished, navigate to Admin > Repair > Quick Repair and Rebuild. This will rebuild the language files for your display text.

Last Modified: 03/09/2016 03:41pm

Font Manager

Overview

The FontManager class is a stand-alone class that manages all the fonts for TCPDF.

Font Cache

The font list is built by the font manager with the listFontFiles() or getSelectFontList() methods. The list is then saved in the cache as ./cache/Sugarpdf/cachedFontList.php. This caching process prevents unnecessary parsing of the fonts folder. The font cache is automatically cleared when the delete() or add() methods are used. When you create a Font loadable module you will have to call the clearCachedFile() method in a post_execute and post_uninstall action in the manifest.php to clear the font cache.

Font Framework

The stock fonts for TCPDF are stored in the directory ./include/tcpdf/fonts/. If you would like to add additional fonts to the system, they should be added to the ./custom/include/tcpdf/fonts/ directory.

Last Modified: 11/12/2015 11:04pm

Teams

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/10/2015 04:35pm

Introduction

Overview

Overview of the team framework.

Teams

Teams provide the ability to assign a record to multiple teams that will limit access at the record level. This provides more flexibility in sharing data across functional groups and is only applicable to paid versions of sugar.

Database Tables

Table Name	Description
teams	Each record in this table represents a team in the system.
team_sets	Each record in this table represents a unique team combination. For example, each user's private team will have a

	corresponding team set entry in this table. A team set may also be comprised of one or more teams.
team_sets_teams	The team_sets_teams table maintains the relationships to determine which teams belong to a team set. Each table that previously used the team_id column to maintain team security now uses the team_set_id column's value to associate the record to team(s).
team_sets_modules	This table is used to manage team sets and keeps track of which modules have records that are or were associated to a particular team set.

It is very important that you do not modify these tables without going through the PHP object methods. Manipulating the team sets with direct SQL may cause undesired side effects in the system due to how the validations and security methods work.

Module Team Fields

In addition to the team tables, each module table also contains a team_id and team_set_id column. The team_set_id contains the id of a record in the team_sets table that contains the unique combination of teams associated to the record. The team_id will contain the id of the primary team designated for a record.

Team Sets (team_set_id)

As mentioned above, Sugar implemented this feature not as a many-to-many relationship but as a one-to-many relationship. On each table that had a 'team_id' field, we added a 'team_set_id' field. We have also added a 'team_sets' table which maintains the team_set_id, a 'team_sets_teams' table which relates a team set to the teams. When performing a team based query we use the 'team_set_id' field on the module table to join to 'team_sets_teams.team_set_id' and then join all of the teams associated with that set. Given the list of teams from team_memberships, we can then decide if the user has access to the record.

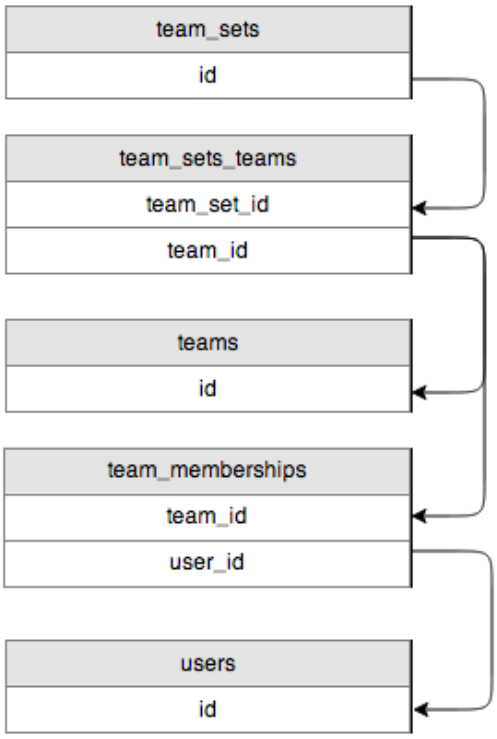
Primary Team (team_id)

The 'team_id' is still being used, not only to support backwards compatibility with workflow and reports but also to provide some additional features. When

displaying a list we use the team set to determine whether the user has access to the record, but when displaying the data, we show the team from team id in the list. When the user performs a mouse over on that team Sugar performs an Ajax call to display all of the teams associated with the record. This 'team_id' field is designated as the Primary Team because it is the first team shown in the list, and for sales territory management purposes, can designate the team that actually owns the record and can report on it.

Team Security

The team_sets_teams table allows the system to check for permissions on multiple teams. The following diagram illustrates table relationships in SugarBean's add_team_security_where_clause method.



Using the team_sets_teams table, the system will determine which teams are associated with the team_set_id and then look in the team_memberships table for users that belong to the team(s).

TeamSetLink

Typically any relationship in a class is handled by the data/Link2.php class. As a part of Dynamic Teams, we introduced the ability to provide your own custom Link

class to handle some of the functionality related to managing relationships. The team_security parent vardefs in the sugar objects contains the following in the 'teams' field definition:

```
'link_class' => 'TeamSetLink','link_file' =>
'modules/Teams/TeamSetLink.php',
```

The link_class entry defines the class name we are using and the link_file tells us where that class file is located. This class extends the legacy Link.php and overrides some of the methods used to handle relationships such as 'add' and 'delete'.

Last Modified: 01/15/2016 10:01pm

Manipulating Teams Programmatically

Overview

How to manipulate team relationships.

Fetching Teams

To fetch teams related to a bean, you will need to retrieve an instance of a TeamSet object and use the getTeams() method to retrieve the teams using the team_set_id. An example is shown below:

```
//Create a TeamSet bean - no BeanFactory
require_once('modules/Teams/TeamSet.php');
$teamSetBean = new TeamSet();

//Retrieve the bean
$bean = BeanFactory::getBean($module, $record_id);

//Retrieve the teams from the team_set_id
$teams = $teamSetBean->getTeams($bean->team_set_id);
```

Adding Teams

To add a team to a bean, you will need to load the teams relationship and use the add() method. This method accepts an array of team ids to add. An example is shown below:

```
//Retrieve the bean
$bean = BeanFactory::getBean($module, $record_id);

//Load the team relationship
$bean->load_relationship('teams');

//Add the teams
$bean->teams->add(
    array(
        $team_id_1,
        $team_id_2
    )
);
```

Considerations

- If adding teams in a logic hook, the recommended approach is to use an after_save hook rather than a before_save hook as the $_REQUEST may reset any changes you make.

Removing Teams

To remove a team from a bean, you will need to load the teams relationship and use the remove() method. This method accepts an array of team ids to remove. An example is shown below:

```
//Retrieve the bean
$bean = BeanFactory::getBean($module, $record_id);

//Load the team relationship
$bean->load_relationship('teams');

//Remove the teams
$bean->teams->remove(
    array(
        $team_id_1,
        $team_id_2
    )
);
```

Considerations

- If removing teams in a logic hook, the recommended approach is to use an after_save hook rather than a before_save hook as the $_REQUEST may reset any changes you make.

Replacing Team Sets

To replace all of the teams related to a bean, you will need to load the teams relationship and use the replace() method. This method accepts an array of team ids. An example is shown below:

```
//Retrieve the bean
$bean = BeanFactory::getBean($module, $record_id);

//Load the team relationship
$bean->load_relationship('teams');

//Set the primary team
$bean->team_id = $team_id_1

//Replace the teams
$bean->teams->replace(
    array(
        $team_id_1,
        $team_id_2
    )
);

//Save to update primary team
$bean->save()
```

Considerations

- If replacing teams in a logic hook, the recommended approach is to use an after_save hook rather than a before_save hook as the $_REQUEST or workflow may reset any changes you make.
- This method does not replace (or set) the primary team for the record. When replacing teams, you need to also make sure that the primary team, determined by the team_id field, is set appropriately and included in the replacement ids. If this is being done in a logic hook you should set the primary team in a before_save hook and replace the team set in the after_save hook.

Function Examples:

```
//before save function
function before_save_hook($bean, $event, $arguments)
{
    $bean->team_id = $team_id_1;
}

//after save function
function after_save_hook($bean, $event, $arguments)
{
    $bean->teams->replace(
        array(
            $team_id_1,
            $team_id_2
        )
    );
}
```

Creating and Retrieving Team Set IDs

To create or retrieve the team_set_id for a group of teams, you will need to retrieve an instance of a TeamSet object and use the addTeams() method. If a team set does not exist, this method will create it and return an id. An example is below:

```
//Create a TeamSet bean - no BeanFactory
require_once('modules/Teams/TeamSet.php');
$teamSetBean = new TeamSet();

//Retrieve/create the team_set_id
$team_set_id = $teamSetBean->addTeams(
    array(
        $team_id_1,
        $team_id_2
    )
);
```

Last Modified: 11/10/2015 04:43pm

Themes

Overview

An overview of the theme framework

Themes

The goal of the Themes framework is to reduce the complexity of the current themes and the amount of work needed to create new themes in the product. The framework also provides tools for easily changing a theme in an upgrade-safe way without modifying the theme directly. This is all possible through an inheritance mechanism that is part of the framework, where themes can build upon other themes to build the user interface for the application. This directly reduces the duplication of code in the application, and enables new UI elements to be supported more easily in any theme.

Theme Directory Structure

The theme directory has a more formal structure to it, which must be followed in order for the inheritance mechanism in the themes framework to work correctly. It is as follows:

- ./themes/
- ./themes/<theme name>/
- ./themes/<theme name>/themedef.php
- ./themes/<theme name>/css/ (all css files go here)
- ./themes/<theme name>/css/style.css
- ./themes/<theme name>/css/print.css
- ./themes/<theme name>/images/ (all images go here)
- ./themes/<theme name>/js/ (all js files go here)

The themedef.php file specified also has a specific format to it so that it can be interpreted properly by the application. It is as follows:

```
$themedef = array(

    // theme name
    'name' => 'Sugar',

     // short description of the theme
    'description' => 'Sugar',

     // maximum number of tabs shown in the bar
    'maxTabs' => $max_tabs,

     // true if png image files are used in this theme, false if gifs
    'pngSupport' => true,
```

```
     // name of the theme this theme inherits from, if something other
than the default theme.
    'parentTheme' => 'ParentTheme',
    'barChartColors' => array(....),
    'pieChartColors' => array(....),
);
```
Please note that only the 'name' specification is required; all other elements are optional.

Theme Development

When you are developing a new theme or adding modifications to an existing theme, it is recommended to turn developer mode on in the 'System Settings' in the 'Admin' section of the application. This is so that any changes you make will become immediately visible to you, making testing you changes easier. Once the theme development is complete, it is recommended that you turn off the Developer Mode.
The theme framework performance is greatly enhanced with the use of an external caching mechanism such as APC or Memcache. It is highly recommended to have this in place.

Changing a Theme

Modifications to any theme in the application can be made in the custom/themes/ directory under the theme in question. For example, to replace the default Accounts.gif image in the Sugar theme, drop the new.gif image file into the custom/themes/<theme name>/images/ directory. You can do the same for CSS and JS files; in these cases the given files will be appended to the existing ones instead of being used in place of them.
The order in which directories are searched for overriding images/css/js files is as follows:

1. ./custom/themes/<theme name>/
2. ./themes/<theme name>/
3. any parent themes (custom directory first, then regular one)
4. ./custom/themes/default/
5. ./themes/default/

Creating a New Theme

The easiest way to create a new theme is to find a base themes that you like and base your design on it. This reduces the amount of CSS work you'll have to do on your own, and you can rest assured that any theme fixes will also be fixed in your theme, thanks to inheritance. To do this, you'll need to create a themedef.php file

with the following specifications in it:

```
$themedef = array(
    // theme name
    'name' => "MySugar",

     // optional, short description of the theme
    'description' => "Sugar theme for me",

     // name of the theme this theme inherits from, in this case the
Sugar theme
    'parentTheme' => "Sugar",
);
```

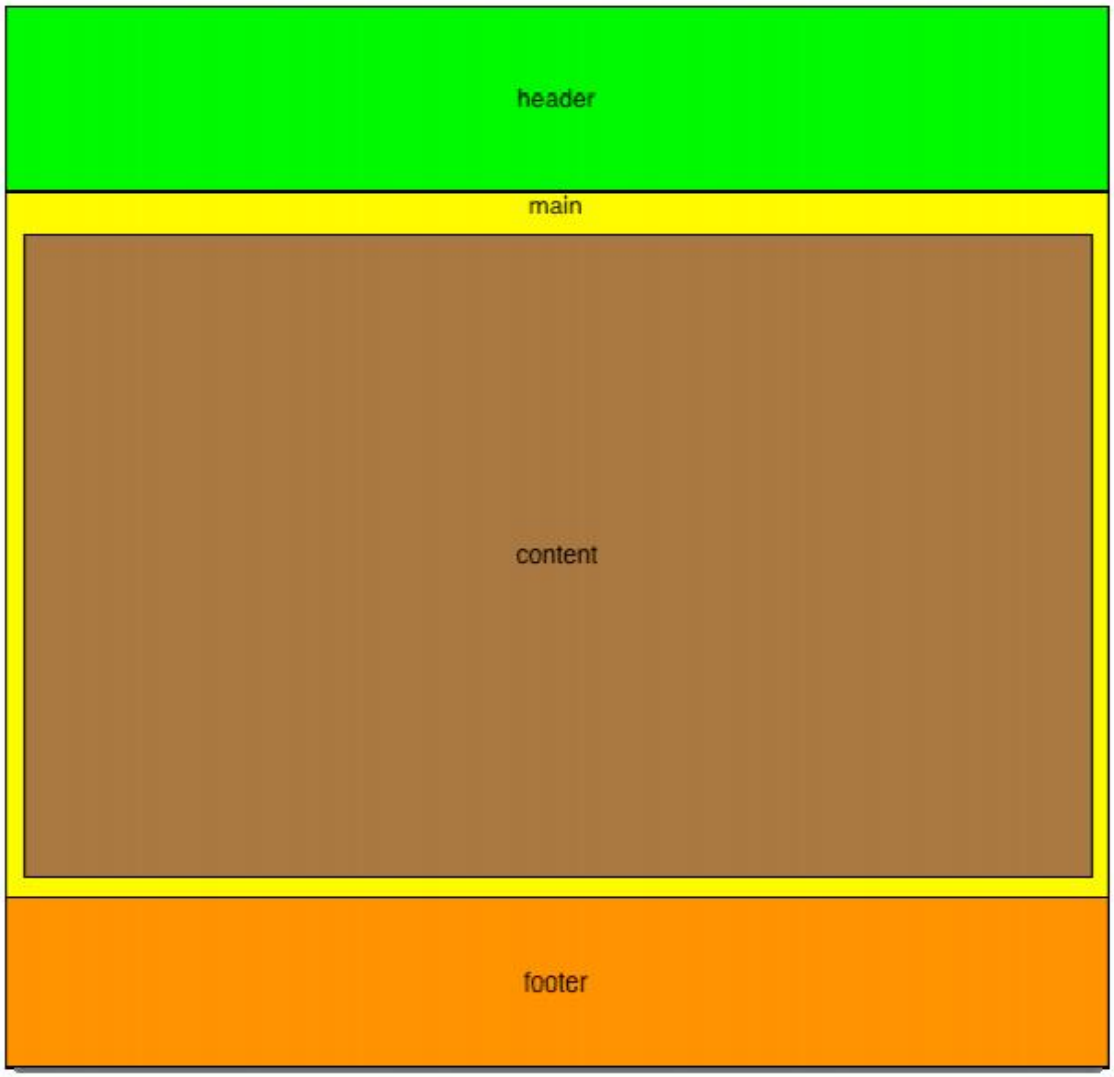
You can now add any CSS, images, and/or JS code to their respective directories to make the needed alterations to the parent theme to create the desired theme. It is by no means a requirement to derive a theme from one of the existing ones in the product; if you do not specify the 'parentTheme' attribute above, then the theme settings in the themes/default/ directory will be used.

Element Reference Guide

Below is a guide of the various id elements and classes that are used in the application. IDs are indicated below where they are prefixed by a # sign (i.e. #dog for the ID dog), and classes are specified by a '.' (.cat for class cat).

.edit	EditView containers
.detail	DetailView containers
.list	ListView containers
.view	Styles for any of the detail, edit. list, and search view containers
.search	Search container

The below figure illustrates the where the main UI div elements are located and what their IDs are.



Packaging Custom Themes

The Upgrade Wizard, accessible through the Admin screen, allows you to apply themes without unzipping the files manually. The theme is also actually added to the "Theme" dropdown on the Sugar Login screen.
The Upgrade Wizard relies on a file named manifest.php, which should reside alongside the root directory of your theme ZIP file.
The following section outlines the format of the manifest file. An example manifest file can be found in the following section:

- acceptable_sugar_flavors : Contains which Sugar editions the package can be installed on. Accepted values are any combination of: CE, PRO, CORP, ENT, and ULT.
- acceptable_sugar_versions : This directive contains two arrays:
    - exact_matches : each element in this array should be one exact version

string, i.e. "6.0.0b" or "6.1.0"

    - regex_matches : each element in this array should be one regular expression designed to match a group of versions, i.e. "6\\.1\\.0[a-z]"

  - author : Contains the author of the package; for example, "SugarCRM Inc."
- copy_files : An array detailing the source and destination of files that should be copied during installation of the package. See the example manifest file below for details.
- description : A description of the package. Displayed during installation.
- icon : A path (within the package ZIP file) to an icon image that will be displayed during installation. Examples include "./patch_directory/icon.gif" and "./patch_directory/images/theme.gif"
- is_uninstallable : Setting this directive to TRUE allows the Sugar administrator to uninstall the package. Setting this directive to FALSE disables the uninstall feature.
- name : The name of the package. Displayed during installation.
- published_date : The date the package was published. Displayed during installation.
- type : The package type. This should be set to "theme"
- version : The version of the patch, i.e. "1.0" or "0.96-pre1"

Example Theme Manifest File

The following is an example manifest.php file:

```
<?php

    $manifest = array (
        'acceptable_sugar_versions' => array (
            'exact_matches' => array (),
            'regex_matches' => array (
                0 => '3.5.[01][a-z]?'
            ),
        ),

        'acceptable_sugar_flavors' => array (
            0 => 'OS',
            1 => 'PRO',
            2 => 'ENT',
        ),

        'name' => 'Theme Name',
        'description' => 'Theme Description',
        'author' => 'SugarCRM Inc.',
        'published_date' => '2005-09-15 16:00:00',
```

```
        'version' => '3.5.1',
        'type' => 'theme',
        'is_uninstallable' => TRUE,
        'icon' => 'ThemeName/images/Themes.gif',
        'copy_files' => array (
            'from_dir' => 'ThemeName',
            'to_dir' => 'themes/ThemeName',
            'force_copy' => array (),
        ),
    );
?>
```

You'll need to create a root directory that contains the theme directory. The name of this root directory (ThemeName) is what should be used in the from_dir element of the copy_files array in manifest.php. You'll also need to place your manifest.php alongside this root directory. Create a ZIP file containing the root directory and the manifest.php file at the top level. Now your language pack is ready to be installed with the Upgrade Wizard.

Tips

- Pick your canvas : Think about how you want the general look and feel of your theme, and see which out-of-the-box existing Sugar theme best fits. This will reduce any extra work that may come out of rearranging the layout because you chose the first theme you saw.
- Replace All : When changing colors in the css files, do a "replace all" on the file. For example, if you are changing the color '#FF0000' to '#CC0000', change all instances of '#FF0000' to '#CC0000'. Eventually you will get to a point where you may want your changes to only affect one class, and may have to go back and refine some of the mass changes. However doing this initially will usually yield favorable results, and save you some time.
- Check your work : So you have all your css laid out and your home page looks good? Did you check your Edit and List views? Did you check the calendar popup widgets (from date fields)? Often, developers forget to check the css for Sugar Dashlets, reports and charts. Do not forget to do a thorough check, and keep in mind that you may have to tweak with navigation.css, layout_utils.php, and sugarColors.xml files before being finally done.
- Personalize your theme : Remember this is your theme now. If you want to add a new div, or introduce a new class, you do not have to make it fit within the confines of the theme with which you started. Most of the images are transparent and work fine, but changing the look and feel of those would add an extra degree of customization. So, go ahead and add your flash intro, embedded mp3 or Star Wars Background.

Last Modified: 10/05/2015 12:14am

TinyMCE

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 02:52am

Modifying the TinyMCE Editor

Overview

This article is a brief overview on how to modify the default settings for the TinyMCE Editor.

TinyMCE Editor

The default settings for TinyMCE can be configured by creating an override file. There are two different overrides that can be applied to buttons and default settings.

Overriding Buttons

The first override file is for the toolbar buttons. To do this, you must create ./custom/include/tinyButtonConfig.php. Within this file, you will be able to override the button layout for the TinyMCE editor.

There are 3 different layouts you can change:

default

This is used when creating an email template.

email_compose

This is used when composing an email from the full form under the emails module.

email_compose_light

This is used when doing a quick compose from a listview or subpanel.

Example File

./custom/include/tinyButtonConfig.php

```
<?php
    //create email template
    $buttonConfigs['default'] = array(
        'buttonConfig' =>
"code,help,separator,bold,italic,underline,strikethrough,separator,jus
tifyleft,justifycenter,justifyright,justifyfull,separator,forecolor,ba
ckcolor,separator,styleselect,formatselect,fontselect,fontsizeselect,"
,
        'buttonConfig2' =>
"cut,copy,paste,pastetext,pasteword,selectall,separator,search,replace
,separator,bullist,numlist,separator,outdent,indent,separator,ltr,rtl,
separator,undo,redo,separator,
link,unlink,anchor,image,separator,sub,sup,separator,charmap,visualaid
",
        'buttonConfig3' =>
"tablecontrols,separator,advhr,hr,removeformat,separator,insertdate,in
serttime,separator,preview"
    );
    //Standard email compose
    $buttonConfigs['email_compose'] = array(
        'buttonConfig' =>
"code,help,separator,bold,italic,underline,strikethrough,separator,bul
list,numlist,separator,justifyleft,justifycenter,justifyright,justifyf
ull,separator,forecolor,backcolor,separator,styleselect,formatselect,f
ontselect,fontsizeselect,",
        'buttonConfig2' => "", //empty
        'buttonConfig3' => "" //empty
    );
    //Quick email compose
    $buttonConfigs['email_compose_light'] = array(
        'buttonConfig' =>
"code,help,separator,bold,italic,underline,strikethrough,separator,bul
list,numlist,separator,justifyleft,justifycenter,justifyright,justifyf
ull,separator,forecolor,backcolor,separator,styleselect,formatselect,f
ontselect,fontsizeselect,",
        'buttonConfig2' => "", //empty
```

```
        'buttonConfig3' => "" //empty
    );
```

Overriding Default Settings

The second override file is for basic TinyMCE functionality. To do this, you must create ./custom/include/tinyMCEDefaultConfig.php. TinyMCE has quite a few settings that can be altered, so the best reference for configuration settings can be found in the TinyMCE Configuration Documentation.

Example File

./custom/include/tinyMCEDefaultConfig.php

```
<?php
    $defaultConfig = array(
        'convert_urls' => false,
        'valid_children' => '+body[style]',
        'height' => 300,
        'width' => '100%',
        'theme' => 'advanced',
        'theme_advanced_toolbar_align' => "left",
        'theme_advanced_toolbar_location' => "top",
        'theme_advanced_buttons1' => "",
        'theme_advanced_buttons2' => "",
        'theme_advanced_buttons3' => "",
        'strict_loading_mode' => true,
        'mode' => 'exact',
        'language' => 'en',
        'plugins' =>
'advhr,insertdatetime,table,preview,paste,searchreplace,directionality
',
        'elements' => '',
        'extended_valid_elements' =>
'style[dir|lang|media|title|type],hr[class|width|size|noshade],@[class
|style]',
        'content_css' =>
'include/javascript/tiny_mce/themes/advanced/skins/default/content.css
',
    );
```

Creating Plugins

Another alternative to customizing the TinyMCE is to create a plugin. Your plugins will be stored in ./include/javascript/tiny_mce/plugins/. You can find more information on creating plugins here: http://www.tinymce.com/wiki.php/Creating_a_plugin

Last Modified: 09/26/2015 04:20pm

Uploads

This guide provides an overview of uploads.

Last Modified: 11/19/2015 02:52am

Introduction

Overview

The upload directory is used to store files uploaded for note attachments, documents, and module loadable packages.

Uploads

The upload directory is used to store any files uploaded to Sugar. By default, anything uploaded to Sugar is stored in the ./upload/ directory. You can change this directory by updating the upload_dir configuration variable. Once uploaded, the file will be stored in this directory with a GUID name.

You should note that when uploading files to Sugar, there are three file size limits to configure. The first two limits are your PHP upload_max_filesize and post_max_size which are configured in your php.ini. The second limit is the sugar configuration setting for upload_maxsize , which will restrict the upload limit from within Sugar. This setting can also be modified in the application via Admin > System Settings.The smallest of these three values will be honored when an oversized file has been uploaded.

You should note that the PHP-defined file size settings and the upload directory cannot be modified for instances hosted on On-Demand.

Upload Extensions

By default, several extension types are restricted due to security issues. Any files uploaded with these extensions will have '.txt' appended to it. The restricted extensions are listed below:

- php
- php3
- php4
- php5
- pl
- cgi
- py
- asp
- cfm
- js
- vbs
- html
- htm

You can add or remove extensions to this list by modifying sugar configuration setting for upload_badext . You should note that this setting cannot be modified for instances hosted on On-Demand.

How Files Are Stored

Note Attachments

When a file is uploaded to Sugar attached to a note, the file will be moved to the upload directory with a GUID name matching that of the notes id. The attributes of the file, such as filename and file_mime_type, will be stored in the note record.

The SQL to fetch information about a notes attachment is shown below:

```
SELECT filename, file_mime_type FROM notes;
```

Email Attachments

Email attachments are stored the same way as note attachments. When an email is imported to Sugar, the file will be moved to the upload directory with a GUID file

name matching that of the notes id. The attributes of the file, such as filename and file_mime_type, will be stored in the note record and the note will have a parent_type of 'Emails'. This relates the attachment to the emails content.

The SQL to fetch information about an emails attachment is shown below:

```
SELECT filename, file_mime_type FROM notes INNER JOIN emails ON
notes.parent_type = 'Emails' AND notes.parent_id = emails.id INNER
JOIN emails_text ON emails.id = emails_text.email_id;
```

Picture Fields

Picture fields will upload the image to the upload directory with a GUID name and store the GUID in the database field on the record. An example of picture field can be found on the Contacts module.

For a contact, the id of the picture attachment can be found with the SQL below:

```
SELECT picture FROM contacts;
```

Document Attachments

Files uploaded to Sugar as documents will be moved to the upload directory with a GUID name matching the document revision id. The document revision id can be found in the document_revision_id field on the documents table. This field maps to the id field on the document_revisions table. The attributes of the file, such as filename and file_mime_type, are stored in the document revision record.

The SQL to fetch information about a documents attachment is shown below:

```
SELECT filename, file_mime_type, file_ext FROM document_revisions
INNER JOIN documents ON documents.id = document_revisions.document_id;
```

Knowledge Base Attachments

When working with the Knowledge Base, files and images attached to the form before the record is saved are uploaded and stored in the upload directory with the name <GUID><File Name> using the KbdocAttachments action found in ./modules/KBDocuments/KbdocAttachments.php. Once the record is saved, these files are copied and saved in the upload directory with a GUID name matching that of the document revision id.

The SQL to fetch information about a knowledge base attachment is shown below:

```
SELECT document_revisions.filename,

document_revisions.file_mime_type,

document_revisions.file_ext FROM document_revisions INNER JOIN
kbdocument_revisions ON document_revisions.id =
kbdocument_revisions.document_revision_id INNER JOIN kbdocuments ON
kbdocument_revisions.kbdocument_id = kbdocuments.id;
```

Module Loadable Packages

Module Loader packages are stored in the system differently than other uploads. They are uploaded to the ./upload/upgrades/module/ directory with their original file name. The details of the package, such as installation status and description, are stored in the upgrade_history table.

The SQL to fetch information about an installed package is show below:

```
SELECT * FROM upgrade_history;
```

CSV Imports

When importing records into Sugar, the most recent uploaded CSV file is stored in the upload directory as IMPORT_<module>_<user id>. Once the import has been run, the results of the import are stored in ./upload/import/ directory using a predefined format using the current users id. The files created will be as follows:

- dupes_<user id>.csv : The list of duplicate records found during the import.
- dupesdisplay_<user_id>.csv : The HTML formatted CSV for display to the user after import.
- error_<user id>.csv : The list of errors encountered during the import.
- errorrecords_<user_id>.csv : The HTML formatted CSV for display to the user after import.
- errorrecordsonly_<user id>.csv : The list of records that encountered an error.
- status_<user id>.csv : Determines the status of the users import.

Last Modified: 01/15/2016 09:56pm

Web Services

Sugar provides a Web Services API interface for developers to build integrations with Sugar for reading and writing data. Sugar provides Web Services APIs through the NuSOAP PHP implementation of the SOAP and REST protocol. SOAP (Simple Object Access Protocol) is used for making Remote Procedure Calls through the HTTP protocol by relaying messages in XML. The SugarSoap APIs, built on top of the NuSOAP PHP library, are included in the Sugar Community, Sugar Professional, Sugar Corporate, Sugar Enterprise, and Sugar Ultimate editions. REST (Representational State Transfer) is used for making method calls through HTTP protocol by sending and receiving messages in JSON/Serialize format. Framework supports the addition of multiple formats for REST. For example, you can add XML format to send and receive data.

Last Modified: 09/26/2015 04:20pm

REST

Overview

REST service documentation.

What is REST?

REST stands for 'Representational State Transfer'. This protocol is used by Sugar to exchange information both internally and externally.

How do I access the REST service?

The REST service in SugarCRM and be found by navigating to:

```
http://{sugar_url)/service/{version}/rest.php
```

Where 'sugar_url' is the url of your Sugar instance and 'version' is the latest version of the API specific to your release of Sugar. You can find out more about versioning in the section titled 'API: Versioning'.

Input / Output Datatypes

The default input / output datatype for REST is JSON / PHP serialize.

These datatype files, SugarRestJSON.php and SugarRestSerialize.php, are in:

```
service/core/REST/
```

Defining your own datatypes

You can also define you own datatype. To do this, you need to create a new file such as:

```
service/core/REST/SugarRest<CustomDataType>.php
```

Next, you will need to override generateResponse() and serve() functions. The Serve function decodes or unserializes the appropriate datatype based on the input type; the generateResponse function encodes or serializes it based on the output type.

See service/test.html for more examples on usage. In this file, the getRequestData function, which generates URL with json, is both the input_type and the response_type. That is, the request data from the javascript to the server is json and response data from server is also json. You can mix and match any datatype as input and output. For example, you can have json as the input_type and serialize as the response_type based on your application's requirements.

REST Requests

If you are making a REST request via cURL from within the Sugar application, you can make use of the SugarHttpClient Class. This class will help automate your cURL requests for you.

REST Failure Response

If a call failure should occur, the result will be as shown below:

Name	Type	Description
name	String	Error message.
number	Integer	Error number.
description	String	Description of error.

Last Modified: 01/15/2016 09:56pm

SOAP

Overview

SOAP service documentation.

What is SOAP?

SOAP stands for 'Simple Object Access Protocol'. SOAP is a simple XML-based protocol that is used to allow applications to exchange information.

How do I access the SOAP service?

The SOAP service in SugarCRM and be found by navigating to:

```
http://{sugar_url)/service/{version}/soap.php
```

Where 'sugar_url' is the url of your Sugar instance and 'version' is the latest version of the API specific to your release of Sugar. You can find out more about versioning in the section titled 'API: Versioning'. The default WSDL is formatted as rpc/encoded.

WS-I 1.0 Compliancy

Sugar supports generating a URL that is WS-I compliant. When accessing the soap entry point, you can access the WSDL at:

```
http://{sugar_url)/service/{version}/soap.php?wsdl
```

By default, the WSDL is formatted as rpc/encoded, however, this can be changed by specifying a 'style' and 'use' url-paramater. An example of this is:

```
http://{sugar_url)/service/{version}/soap.php?wsdl&style=rpc&use=literal
```

URL Parameters

style

- rpc
- document

use

- encoded
- literal

Validation

This WSDL (rpc/literal) was successfully verified against Apache CXF 2.2.

SOAP Failure Response

If a call failure should occur, the result will be as shown below:

Name	Type	Description
faultcode	Integer	Fault ID.
faultactor	String	Provides information about what caused the fault to happen.
faultstring	String	Fault Message.
detail	String	Description of fault.

Last Modified: 01/15/2016 10:01pm

NuSOAP

Overview

NuSOAP is a SOAP Toolkit for PHP that doesn't require PHP extensions.

Where Can I Get It?

NuSOAP can be downloaded from http://nusoap.sourceforge.net

How Do I Use It?

After you have downloaded NuSOAP, you will need to extract the zip file contents to a storage directory. Once extracted, you will reference "lib/nusoap.php" in your PHP SOAP application.

Example

```
<?php

//require NuSOAP
require_once("./lib/nusoap.php");

//retrieve WSDL
$client = new
nusoap_client("http://{site_url}/service/v4/soap.php?wsdl", 'wsdl');
```

Last Modified: 09/26/2015 04:23pm

SOAP vs. REST

Overview

Information regarding SOAP and REST.

Methods

There are various methods that are only contained within the REST API. A list of examples are:

- get_module_layout
- get_module_layout_md5
- get_quotes_pdf method
- get_report_pdf method
- snip_import_emails
- snip_update_contacts
- job_queue_cycle
- job_queue_next
- job_queue_run
- oauth_access method
- oauth_access_token
- oauth_request_token

When using a method, you should first verify that it is available for your preferred API.

Technology

There are significant differences between how the REST and SOAP protocols function on an implementation level (e.g. Performance, response size, etc). Deciding which protocol to use is up to the individual developer and is beyond the scope of this guide. Starting in SugarCRM version 6.2.0, there are some deviations between the protocols with the v4 API. There are additional core calls that are only made available through the REST protocol. The calls are documented below in the 'Method Calls' section.

Last Modified: 11/19/2015 02:52am

Versioning

What is API versioning?

As of version 5.5.0, when adding new functionality to web services, we release a new version of the web service to prevent any changes from causing issues for customers already interacting with an existing service to help extend the application in an upgrade-safe manner.

Which version should I be using?

The easiest way to figure out which version you should be using is to navigate to

your service directory. This directory will be located in your sugar installations root path '/service/'.

In this directory, you will see a list of directories prefixed with a "v". A few examples are: v2, v3 and v4. The higher the number, the more recent the version.

*Please note that in 6.7.x, the v10 API is not currently supported for any type of custom development.

Version Quick Reference

Release	Version	REST URL	SOAP URL
5.2.x	v1	Not Available	http://{site_url}/soap.php
5.5.x	v2	http://{site_url}/service/v2/rest.php	http://{site_url}/service/v2/soap.php
6.0.x	v2	http://{site_url}/service/v2/rest.php	http://{site_url}/service/v2/soap.php
6.1.x	v3_1	http://{site_url}/service/v3_1/rest.php	http://{site_url}/service/v3_1/soap.php
6.2.x	v4	http://{site_url}/service/v4/rest.php	http://{site_url}/service/v4/soap.php
6.3.x	v4	http://{site_url}/service/v4/rest.php	http://{site_url}/service/v4/soap.php
6.4.x	v4_1	http://{site_url}/service/v4_1/rest.php	http://{site_url}/service/v4_1/soap.php
6.5.x	v4_1	http://{site_url}/service/v4_1/rest.php	http://{site_url}/service/v4_1/soap.php
6.6.x	v4_1	http://{site_url}/service/v4_1/rest.php	http://{site_url}/service/v4_1/soap.php
6.7.x	v4_1	http://{site_url}/service/v4_1/rest.php	http://{site_url}/service/v4_1/soap.php

Last Modified: 07/27/2016 04:57pm

Method Calls

The various methods of the v4_1 API.

Last Modified: 09/26/2015 04:23pm

get_available_modules

Overview

Retrieves a list of available modules in the system.

Available APIs

- SOAP
- REST

Definition

get_available_modules(session, filter)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
filter	String	String to filter the modules with. Possible values are 'default', 'mobile', 'all'.

Result

Name	Type	Description
result	new_module_fields | Array	The call result.
result.modules	Array	The list of available modules.

Name	Type	Description
result.modules[].module_key	String	The modules key.
result.modules[].module_label	String	The display label for the module.
result.modules[].favorite_enabled	String	Whether favorites are enabled for the module.
result.modules[].acls	Array	The ACL list for the module

Change Log

Version	Change
v3	Added filter parameter. Accepts the values 'default', 'mobile', 'all'.

PHP

```
$get_available_modules_parameters = array(
    //Session id
    "session" => $session_id,

    //Module filter. Possible values are 'default', 'mobile', 'all'.
    "filter" => 'all',);
```

Last Modified: 09/26/2015 04:23pm

get_document_revision

Overview

Retrieves a specific document revision.

Available APIs

- SOAP
- REST

Definition

get_document_revision(session, i)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
i	String	The ID of the document revision.

Result

Name	Type	Description
result	new_return_document_revision| Array	The call result.
result.document_revision	Array	The details of the document revision.
result.document_revision.id	String	The document ID.
result.document_revision.document_name	String	The document name.
result.document_revision.revision	String	The document revision number
result.document_revision.filename	String	The filename of the file.
result.document_revision.file	String	The binary contents of the file.

Change Log

Version	Change
v2	Return type was changed from return_document_revision to new_return_document_revision.

PHP

```
$get_document_revision_parameters = array(
    //Session id
    "session" => $session_id,

    //The attachment details
    "i" => '723b7dcb-27b3-e53d-b348-50bd283f8e48',);
```

Last Modified: 09/26/2015 04:23pm

get_entries

Overview

Retrieves a list of beans based on specified record IDs.

Available APIs

- SOAP
- REST

Definition

get_entries(session, module_name, ids, select_fields, link_name_to_fields_array, track_view)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
module_name	String	The name of the module from which to retrieve records. Note: This is the modules key which may not be the same as the modules display name.
ids	String	The list of record IDs to retrieve.
select_fields	select_fields | Array	The list of fields to be returned in the results. Specifying an empty array will return all fields.
link_name_to_fields_array	link_names_to_fields_array | Array	A list of link names and the fields to be returned for each link.
track_view	Boolean	Flag the record as a recently viewed item.

Result

Name	Type	Description
result	get_entry_result_version2 | Array	The call result.
result.entry_list	Array	The record's name-value pair for the simple datatypes excluding the link field data. If you do not have access to the object, entry_list[].name_value_list will notify you.
result.relationship_list	Array	The records link field data.

Change Log

Version	Change
v3_1	Added track_view parameter.
v2	Added link_name_to_fields_array parameter.
v2	Return type was changed from get_entry_result to get_entry_result_version2.

PHP

```
$get_entries_parameters = array(
     //session id
     'session' => $session_id,

     //The name of the module from which to retrieve records
     'module_name' => 'Accounts',

     //An array of record IDs
     'ids' => array(
         '14b0c0ca-3ea2-0ee8-f3be-50aa57c11ee7',
     ),
     //The list of fields to be returned in the results
     'select_fields' => array(
        'name',
        'billing_address_state',
        'billing_address_country'
     ),
     //A list of link names and the fields to be returned for each
link name
     'link_name_to_fields_array' => array(
          array(
               'name' => 'email_addresses',
               'value' => array(
                    'email_address',
                    'opt_out',
                    'primary_address'
               ),
          ),
     ),
    //Flag the record as a recently viewed item
    'track_view' => true,
);
```

Last Modified: 09/26/2015 04:23pm

get_entries_count

Overview

Retrieves a list of beans based on query specifications.

Available APIs

- SOAP
- REST

Definition

get_entries_count(session, module_name, query, deleted)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
module_name	String	The name of the module from which to retrieve records. Note: This is the modules key which may not be the same as the modules display name.
query	String	The SQL WHERE clause without the word "where".
deleted	Integer	If deleted records should be included in the results.

Result

Name	Type	Description

Name	Type	Description
result	get_entries_count_result | Array	The call result.
result.result_count	Integer	The count of total records.

Change Log

Version	Change

PHP

```
$get_entries_count_parameters = array(
     //Session id
     'session' => $session_id,

     //The name of the module from which to retrieve records
     'module_name' => 'Accounts',

     //The SQL WHERE clause without the word "where".
     'query' => " accounts.name like '%example text%' ",

     //If deleted records should be included in results.
     'deleted' => false
);
```

Last Modified: 09/26/2015 04:23pm

get_entry

Overview

Retrieves a single bean based on record ID.

Available APIs

- SOAP
- REST

Definition

get_entry(session, module_name, id, select_fields, link_name_to_fields_array, track_view)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
module_name	String	The name of the module from which to retrieve records. Note: This is the modules key which may not be the same as the modules display name.
id	String	The ID of the record to retrieve.
select_fields	select_fields | Array	The list of fields to be returned in the results. Specifying an empty array will return all fields.
link_name_to_fields_array	link_names_to_fields_array | Array	A list of link names and the fields to be returned for each link.
track_view	Boolean	Flag the record as a recently viewed item.

Result

Name	Type	Description
result	get_entry_result_version2 | Array	The call result.

Name	Type	Description
result.entry_list	Array	The record's name-value pair for the simple datatypes excluding the link field data. If you do not have access to the object, entry_list[].name_value_list will notify you.
result.relationship_list	Array	The records link field data.

Change Log

Version	Change
v3_1	Added track_view parameter.
v2	Added link_name_to_fields_array parameter.
v2	Return type was changed from get_entry_result to get_entry_result_version2.

PHP

```
$get_entry_parameters = array(
     //session id
     'session' => $session_id,

     //The name of the module from which to retrieve records
     'module_name' => "Contacts",

     //The ID of the record to retrieve.
     'id' => "18df70e4-1422-8bff-6f5f-50aa571fe4e5",

     //The list of fields to be returned in the results
     'select_fields' => array(
          'id',
          'first_name',
          'last_name',
     ),
     //A list of link names and the fields to be returned for each
link name
     'link_name_to_fields_array' => array(
```

```
          array(
               'name' => 'email_addresses',
               'value' => array(
                    'email_address',
                    'opt_out',
                    'primary_address'
               ),
          ),
     ),
    //Flag the record as a recently viewed item
    'track_view' => true,
);
```

Last Modified: 09/26/2015 04:23pm

get_entry_list

Overview

Retrieves a list of beans based on query specifications.

Available APIs

- SOAP
- REST

Definition

get_entry_list(session, module_name, query, order_by, offset, select_fields, link_name_to_fields_array, max_results, deleted, favorites)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.

Name	Type	Description
module_name	String	The name of the module from which to retrieve records. Note: This is the modules key which may not be the same as the modules display name.
query	String	The SQL WHERE clause without the word "where". You should remember to specify the table name for the fields to avoid any ambiguous column errors.
order_by	String	The SQL ORDER BY clause without the phrase "order by".
offset	Integer	The record offset from which to start.
select_fields	select_fields | Array	The list of fields to be returned in the results. Specifying an empty array will return all fields.
link_name_to_fields_array	link_names_to_fields_array | Array	A list of link names and the fields to be returned for each link.
max_results	Integer	The maximum number of results to return.
deleted	Integer	If deleted records should be included in the results.
favorites	Boolean	If only records marked as favorites should be returned.

Result

Name	Type	Description
result	get_entry_result_version2 | Array	The call result.
result.result_count	Integer	The total number of

Name	Type	Description
		records returned in the call.
result.total_count	Integer	The total number of records.
resultnext_offset	Integer	The next offset to retrieve records.
result.entry_list	Array	The record's name-value pair for the simple datatypes excluding the link field data. If you do not have access to the object, entry_list[].name_value_list will notify you.
result.relationship_list	Array	The records link field data.

Change Log

Version	Change
v3_1	Added favorites parameter.
v2	Added link_name_to_fields_array parameter.
v2	Return type was changed from get_entry_list_result to get_entry_list_result_version2.

PHP

```
$get_entry_list_parameters = array(
     //session id
     'session' => $session_id,

     //The name of the module from which to retrieve records
     'module_name' => 'Leads',

     //The SQL WHERE clause without the word "where".
     'query' => "",

     //The SQL ORDER BY clause without the phrase "order by".
     'order_by' => "",
```

```
     //The record offset from which to start.
     'offset' => 0,

     //A list of fields to include in the results.
     'select_fields' => array(
          'id',
          'name',
          'title',
     ),
     //A list of link names and the fields to be returned for each
link name.
     'link_name_to_fields_array' => array(
         array(
             'name' => 'email_addresses',
             'value' => array(
                 'email_address',
                 'opt_out',
                 'primary_address'
             ),
         ),
     ),
     //The maximum number of results to return.
     'max_results' => 2,

     //If deleted records should be included in results.
     'deleted' => 0,

     //If only records marked as favorites should be returned.
     'favorites' => false,
);
```

Last Modified: 09/26/2015 04:23pm

get_language_definition

Overview

Retrieves the language label strings for the specified modules.

Available APIs

- REST

Definition

get_available_modules(session, modules, md5)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
modules	Array	The list of modules to retrieve language definitions for.
md5	Boolean	Setting this to true will return an MD5 hash of the modules labels strings instead of the label strings themselves.

Result

Name	Type	Description
result	Array	The call result. Contains a list of modules.
result[]	Array or String	The list of language definitions or MD5 hashes. This is dependent on the 'md5' parameter.

Change Log

Version	Change
v3_1	Added get_language_definition method.

PHP

```
$get_language_definition_parameters = array(
    //Session id
    'session' => $session_id,

    //The list of modules
    'modules' => array(
        'Accounts'
    ),
    /Whether to return the results as an MD5 hash
    'md5' => false,
);
```

Last Modified: 09/26/2015 04:23pm

get_last_viewed

Overview

Retrieves a list of recently viewed records by module for the current user.

Available APIs

- SOAP
- REST

Definition

get_last_viewed(session, module_names)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.

Name	Type	Description
module_names	module_names | Array	The list of modules to retrieve last viewed records for.

Result

Name	Type	Description
result	last_viewed_list | Array	The call result. Contains a list of reciently viewed records.
result[].id	Integer	The result ID.
result[].item_id	String	The reciently viewed record ID.
result[].item_summary	String	The name of the reciently viewed record.
result[].module_name	String	The name of the module.
result[].monitor_id	String	The monitor ID from the tracker table.
result[].date_modified	String	The date the record was viewed.

Change Log

Version	Change

PHP

```
$get_last_viewed_parameters = array(
    //Session id
    "session" => $session_id,

    //The name of the modules to retrieve last viewed for
    'module_names' => array(
        'Contacts',
```

```
        'Accounts'
    ),
);
```

Last Modified: 01/15/2016 09:56pm

get_modified_relationships

Overview

Retrieves a list of modified relationships between a specific date range. Helps facilitate sync operations for users.

Available APIs

- SOAP
- REST

Definition

get_modified_relationships(session, module_name, related_module, from_date, to_date, offset, max_results, deleted, module_user_id, select_fields, relationship_name, deletion_date)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
module_name	String	The module key to retrieve relationships against.
related_module	String	The related module key to retrieve records off of. This parameter should always be 'Users'.

Name	Type	Description
from_date	String	Start date in YYYY-MM-DD HH:MM:SS format for the date range.
to_date	String	End date in YYYY-MM-DD HH:MM:SS format for the date range.
offset	Integer	The offset to begin returning records from.
max_results	Integer	The max_results to return.
deleted	Integer	Whether or not to include deleted records. Set to 1 to find deleted records.
module_user_id	String	*This parameter is no longer used and is only present for backward compatibility purposes.
select_fields	select_fields	List of fields to select and return as name/value pairs.
relationship_name	String	The name of the relationship name to search on.
deletion_date	String	Date value in YYYY-MM-DD HH:MM:SS format for filtering on deleted records whose date_modified falls within range.

Result

Name	Type	Description
result	modified_relationship_result | Array	The call result.
result.result_count	Integer	The result count.
result.next_offset	Integer	The next offset to retrieve from.

Name	Type	Description
result.entry_list	Array	List of found records.
result.error	Array	Error Message.
result.error.number	Integer	The error number.
result.error.name	String	The name of the error.
result.error.description	String	The description of the error.

Change Log

Version	Change
v4_1	Added get_modified_relationships method.

Considerations

- The module_name parameter should always be'Users'.

PHP

```
$get_modified_relationships_parameters = array(
    //Session id
    'session' => $session_id,

    //The module key to retrieve relationships against.
    //This parameter should always be 'Users'.
    'module_name' => 'Users',

    //The related module key to retrieve records off of.
    'related_module' => 'Meetings',

    //Start date in YYYY-MM-DD HH:MM:SS format for the date range.
    'from_date' => '2000-01-01 01:01:01',

    //End date in YYYY-MM-DD HH:MM:SS format for the date range
    'to_date' => '2013-01-01 01:01:01',

    //The offset to begin returning records from.
```

```
    'offset' => 0,

    //The max_results to return.
    'max_results' => 5,

    //Whether or not to include deleted records. Set to 1 to find
deleted records.
    'deleted' => 0,

    //This parameter is not used.
    'module_user_id' => '',

    //List of fields to select and return as name/value pairs.
    'select_fields' => array(
    ),

    //The name of the relationship name to search on.
    'relationship_name' => 'meetings_users',

    //Date value in YYYY-MM-DD HH:MM:SS format for filtering on
deleted records.
    'deletion_date' => '2012-01-01 01:01:01'
);
```

Last Modified: 09/26/2015 04:23pm

get_module_fields

Overview

Retrieves the list of field vardefs for a specific module.

Available APIs

- SOAP
- REST

Definition

get_module_fields(session, module_name, fields)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
module_name	String	The name of the module from which to retrieve records. Note: This is the modules key which may not be the same as the modules display name.
fields	select_fields | Array	The list of fields to retrieve. An empty parameter will return all.

Result

Name	Type	Description
result	new_module_fields | Array	The call result.
result.module_name	String	The name of the module.
result.table_name	String	The name of the modules primary table.
result.module_fields	Array	The vardefs for each individual field.

Change Log

Version	Change
v2	Added fields parameter.
v2	Return type was changed from module_fields to new_module_fields.

PHP

```
$get_module_fields_parameters = array(
    //Session id
    "session" => $session_id,

    //The name of the module from which to retrieve fields
    'module_name' => "Contacts",

    //List of specific fields
    'fields' => array(),
);
```

Last Modified: 09/26/2015 04:23pm

get_module_fields_md5

Overview

Retrieves the MD5 hash of the vardefs for the specified modules.

Available APIs

- SOAP
- REST

Definition

get_module_fields_md5(session, module_names)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
module_names	select_fields | Array	The list of modules to retrieve MD5 hashes for.

Result

Name	Type	Description
result	md5_results | Array	The call result. Contains a list of module field hashes. Order is based on the module_names parameter.

Change Log

Version	Change

PHP

```
$get_module_fields_md5_parameters = array(
    //Session id
    "session" => $session_id,

    //The name of the modules to retrieve field hashes for
    'module_names' => array(
        'Contacts',
        'Accounts'
    ),
);
```

Last Modified: 09/26/2015 04:23pm

get_module_layout

Overview

Retrieves the layout metadata for a given module given a specific type and view.

Available APIs

- REST

Definition

get_module_layout(session, modules, types, views, acl_check, md5)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
modules	Array	The list of modules to retrieve layouts for.
types	Array	The types of views requested. Current supported types are 'default' (for application) and 'wireless'.
views	Array	The views requested. Current supported types are 'edit', 'detail', 'list', and 'subpanel'.
acl_check	Boolean	Whether or not to check for ACL access.
md5	Boolean	Setting this to true will return an MD5 hash of the modules layouts instead of the layouts themselves.

Result

Name	Type	Description
result	Array	The call result. Contains a list of modules.

Name	Type	Description
result[][$type]	Array	The list of types requested.
result[][$view]	Array or String	The list of layout views requested or MD5 hashes. This is dependent on the 'md5' parameter.

Change Log

Version	Change
v3_1	Added acl_check parameter.
v3	Added get_module_layout method.

PHP

```
$get_module_layout_parameters = array(
    //Session id
    'session' => $session_id,

    //The list of modules
    'modules' => array(
        'Accounts'
    ),

    //The types of views requested
    'types' => array(
        'default',
    ),

    //The views requested
    'views' => array(
        'edit'
    ),

    //Whether or not to check for ACL access
    'acl_check' => false,

    //Whether to return the results as an MD5 hash
    'md5' => true,
```

```
);
```

Last Modified: 09/26/2015 04:23pm

get_module_layout_md5

Overview

Retrieves the MD5 hash value for a layout given a specific module, type and view.

Available APIs

- REST

Definition

get_module_layout_md5(session, modules, types, views, acl_check)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
modules	Array	The list of modules to retrieve layouts for.
types	Array	The types of views requested. Current supported types are 'default' (for application) and 'wireless'.
views	Array	The views requested. Current supported types are 'edit', 'detail', 'list', and 'subpanel'.
acl_check	Boolean	Whether or not to check

Name	Type	Description
		for ACL access.

Result

Name	Type	Description
result	Array	The call result. Contains a list of modules.
result[][$type]	Array	The list of types requested.
result[][$view]	String	The list of MD5 layout view hashes requested.

Change Log

Version	Change
v3_1	Added acl_check parameter.
v3	Added get_module_layout_md5 method.

PHP

```
$get_module_layout_md5_parameters = array(
    //Session id
    'session' => $session_id,

    //The list of modules
    'modules' => array(
        'Accounts'
    ),

    //The types of views requested
    'types' => array(
        'default',
    ),

    //The views requested
    'views' => array(
```

```
        'edit'
    ),

    //Whether or not to check for ACL access
    'acl_check' => false,
);
```

Last Modified: 09/26/2015 04:23pm

get_note_attachment

Overview

Retrieves an attachment associated with a specific note record.

Available APIs

- SOAP
- REST

Definition

get_note_attachment(session, id)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
id	String	The ID of the note record to associate the attachment to.

Result

Name	Type	Description
result	new_return_note_attachment | Array	The call result.
result.note_attachment	Array	The details of the file attachment.
result.note_attachment.id	String	The ID of the note record / attachment.
result.note_attachment.filename	String	The filename of the attachment.
result.note_attachment.file	String	The binary contents of the file.
result.note_attachment.related_module_id	String	The related parent ID.
result.note_attachment.related_module_name	String	The related parent module.

Change Log

Version	Change
v2	Return type was changed from return_note_attachment to new_return_note_attachment.

PHP

```
$get_note_attachment_parameters = array(
    //Session id
    "session" => $session_id,

    //The ID of the note containing the attachment.
    'id' => "9057784d-de17-4f28-c5f9-50bd0f260a43",
);
```

Last Modified: 09/26/2015 04:23pm

get_quotes_pdf

Overview

Generates a PDF for a given quote.

Available APIs

- REST

Definition

get_quotes_pdf(session, quote_id, pdf_format)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
quote_id	String	The ID of the quote record to generate the PDF for.
pdf_format	String	The pdf type requested. 'Standard' is an example.

Result

Name	Type	Description
result	Array	The call result.
result.file_contents	String	The binary contents of the PDF file.

Change Log

Version	Change
v3_1	Added get_quotes_pdf method.

PHP

```
$get_quotes_pdf_parameters = array(
    //Session id
    'session' => $session_id,

    //The quote to generate the pdf for
    'quote_id' => '490cc844-f83a-9b74-9888-50aa575b517c',

    //The pdf type
    'pdf_format' => 'Standard',
);
```

Last Modified: 09/26/2015 04:23pm

get_relationships

Overview

Retrieves a specific relationship link for a specified record.

Available APIs

- SOAP
- REST

Definition

get_relationships(session, module_name, module_id, link_field_name, related_module_query, related_fields, related_module_link_name_to_fields_array, deleted, order_by, offset, limit)

Parameters

Name	Type	Description

Name	Type	Description
session	String	Session ID returned by a previous login call.
module_name	String	The name of the module from which to retrieve records. Note: This is the modules key which may not be the same as the modules display name.
module_id	String	The ID of the specified module record.
link_field_name	String	The name of the link field for the related module.
related_module_query	String	The list of related record IDs you are relating
related_fields	select_fields | Array	An array specifying relationship fields to populate. An example of this is contact_role between Opportunities and Contacts.
related_module_link_name_to_fields_array	link_names_to_fields_array | Array	For every related record returned, specify link field names to field information.
deleted	Integer	
order_by	String	The SQL ORDER BY clause without the phrase "order by".
offset	Integer	The record offset from which to start.
limit	Integer	The maximum number of results to return.

Result

Name	Type	Description
result	get_entry_result_version2 | Array	The call result.

Name	Type	Description
result.entry_list	Array	The record's name-value pair for the simple datatypes excluding the link field data. If you do not have access to the object, entry_list[].name_value_list will notify you.
result.relationship_list	Array	The records link field data.

Change Log

Version	Change
v3	Added order_by parameter.
v2	Removed related_module parameter.
v2	Added link_field_name parameter.
v2	Added related_fields parameter.
v2	Added related_module_link_name_to_fields_array parameter.
v2	Return type was changed from get_relationships_result to get_entry_result_version2.

PHP

```
$get_relationships_parameters = array(
     //session id
     'session' => $session_id,

     //The name of the module from which to retrieve records.
     'module_name' => 'Accounts',

     //The ID of the specified module bean.
     'module_id' => '9f0c0ceb-c512-7103-9456-50aa5787c3f6',

     //The relationship name of the linked field from which to return
records.
     'link_field_name' => 'opportunities',
```

```
     //The portion of the WHERE clause from the SQL statement used to
find the related items.
     'related_module_query' => " opportunities.name IS NOT NULL ",

     //The related fields to be returned.
     'related_fields' => array(
        'id',
        'name'
     ),

     //For every related bean returned, specify link field names to
field information.
     'related_module_link_name_to_fields_array' => array(
         array(
             'name' => 'contacts',
             'value' => array(
                 'id',
                 'first_name',
                 'last_name',
             ),
         ),
     ),

     //To exclude deleted records
     'deleted'=> 0,

     //order by
     'order_by' => ' opportunities.name ',

     //offset
     'offset' => 0,

     //limit
     'limit' => 200,
);
```

Last Modified: 09/26/2015 04:23pm

get_report_entries

Overview

Retrieves a list of report entries based on specified record IDs.

Available APIs

- SOAP
- REST

Definition

get_report_entries(session, ids, select_fields)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
ids	select_fields | Array	An array of record IDs to retrieve.
select_fields	select_fields | Array	The list of fields to be included in the results.

Result

Name	Type	Description
result	get_entry_result_for_reports | Array	The call result.
result.field_list	Array	The list of selected fields.
result.field_list[].name	String	The name of the report.
result.field_list[].type	String	The type of the report.
result.field_list[].label	String	The label of the report.
result.field_list[].required	Boolean	
result.field_list[].options	Array	
result.entry_list	Array	The list of report results
result.entry_list[].id	String	The result ID. This is not the record ID.

Name	Type	Description
result.entry_list[].module_name	String	The name of the module. Normally contains the value 'Reports'.
result.entry_list[].name_value_list	Array	The name value list of the report results.

Change Log

Version	Change
v2	Method get_report_entries was added.

Considerations

- This method is not available in CE.

PHP

```
$get_report_entries_parameters = array(
     //Session id
     'session' => $session_id,

     //An array of record IDs to retrieve.
     'ids' => array(
        '63f1b905-d206-14cb-cb95-50aa5734815f'
     ),

    //The list of fields to be included in the results.
    'select_fields' => array(
    )
);
```

Last Modified: 09/26/2015 04:23pm

get_report_pdf

Overview

Generates a PDF for a specific report.

Available APIs

- REST

Definition

get_report_pdf(session, report_id)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
report_id	String	The ID of the report record to generate the PDF for.

Result

Name	Type	Description
result	Array	The call result.
result.file_contents	String	The binary contents of the PDF file.

Change Log

Version	Change
v3_1	Added get_report_pdf method.

PHP

```
$get_report_pdf_parameters = array(
    //Session id
    'session' => $session_id,

    //The report to generate the pdf for.
    'report_id' => '68ca4de9-7486-72e1-1a56-50aa5757aaab',
);
```

Last Modified: 09/26/2015 04:23pm

get_server_info

Overview

Retrieves info about the Sugar instance.

Available APIs

- SOAP
- REST

Definition

get_server_info()

Parameters

Name	Type	Description
null	null	No parameters available.

Result

Name	Type	Description
result	get_server_info_result | Array	The call result.
result.flavor	String	The flavor of the instance.
result.version	String	The version of the instance.
result.gmt_time	String	The GMT time of the server.

Change Log

Version	Change
v2	Method get_server_info was added to replace get_server_time, get_server_version and get_sugar_flavor.
v2	Method get_server_time was removed.
v2	Method get_server_version was removed.
v2	Method get_sugar_flavor was removed.

PHP

```
//this method does not have any parameters
$get_server_info_parameters = array();
```

Last Modified: 09/26/2015 04:23pm

get_upcoming_activities

Overview

Retrieves a list of upcoming activities for the current user.

Available APIs

- SOAP
- REST

Definition

get_upcoming_activities(session)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.

Result

Name	Type	Description
result	upcoming_activities_list | Array	The call result. Contains a list of upcoming activity records.
result[].id	Integer	The record ID.
result[].module	String	The activity module.
result[].date_due	String	The date the activity is due.
result[].summary	String	The summary of the activity.

Change Log

Version	Change

PHP

```
$get_upcoming_activities_parameters = array(
    //Session id
    "session" => $session_id,
);
```

Last Modified: 09/26/2015 04:23pm

get_user_id

Overview

Retrieves the id of the user currently logged in.

Available APIs

- SOAP
- REST

Definition

get_user_id(session)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.

Result

Name	Type	Description
id	String	The users ID.

Change Log

Version	Change

PHP

```
$get_user_id_parameters = array(
    //Session id
    "session" => $session_id,
);
```

Last Modified: 09/26/2015 04:23pm

get_user_team_id

Overview

Retrieves the ID of the default team of the user who is logged into the current session.

Available APIs

- SOAP
- REST

Definition

get_user_team_id(session)

Parameters

Name	Type	Description
session	String	Session ID returned by a

Name	Type	Description
		previous login call.

Result

Name	Type	Description
default_team_id	Integer	The ID of the current users default ream

Change Log

Version	Change
v2	Added get_user_team_id method.

PHP

```
$get_user_team_id_parameters = array(
    //Session id
    "session" => $session_id,
);
```

Last Modified: 09/26/2015 04:23pm

job_queue_cycle

Overview

Runs through the scheduler cleanup process and cycles the scheduler jobs.

Available APIs

- REST

Definition

job_queue_cycle(session, clientid)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
clientid	String	The client id calling the application. This parameter is of your choosing for the calling application.

Result

Name	Type	Description
result	Array	The call result.
result.results	String	The cycle result. Returns 'ok' on success.

Change Log

Version	Change
v4	Added job_queue_cycle method.

PHP

```
$job_queue_cycle_parameters = array(
    //Session id
    'session' => $session_id,

    //The ID of the calling application.
    'clientid' => 'MyAppID',
```

```
);
```

Last Modified: 09/26/2015 04:23pm

job_queue_next

Overview

Retrieves the next job from the job queue and marks it as 'In Progress'.

Available APIs

- REST

Definition

job_queue_next(session, clientid)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
clientid	String	The client id calling the application. This parameter is of your choosing for the calling application.

Result

Name	Type	Description
result	Array	The call result.

Name	Type	Description
result.results	String	The next job ID.

Change Log

Version	Change
v4	Added job_queue_next method.

PHP

```
$job_queue_next_parameters = array(
    //Session id
    'session' => $session_id,

    //The ID of the calling application.
    'clientid' => 'MyAppID',
);
```

Last Modified: 09/26/2015 04:23pm

job_queue_run

Overview

Runs the specified job.

Available APIs

- REST

Definition

job_queue_run(session, jobid, clientid)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
jobid	String	The ID of the job to run.
clientid	String	The client id calling the application. This parameter is of your choosing for the calling application.

Result

Name	Type	Description
result	Array	The call result.
result.results	Boolean	The result of the job run.
result.message	String	This is only returned if a failure occurs.

Change Log

Version	Change
v4	Added job_queue_run method.

PHP

```
$job_queue_run_parameters = array(
    //Session id
    'session' => $session_id,

    //The ID of the job to run.
    'jobid' => 'd141efd3-d2c7-8a9c-9c02-50c11b491f16',

    //The ID of the calling application.
```

```
    'clientid' => 'MyAppID',
);
```

Last Modified: 09/26/2015 04:23pm

login

Overview

Logs a user into the Sugar application.

Available APIs

- SOAP
- REST

Definition

login(user_auth, application_name, name_value_list)

Parameters

Name	Type	Description
user_auth	user_auth | Array	Contains the parameters to authenticate a user.
user_auth.user_name	String	The user name of your user
user_auth.password	String	The MD5 hash of the users password.
application name	String	The name of the application logging in.
name_value_list	name_value_list | Array	Sets the name_value pair. The parameter is used to set values for the 'language' and

Name	Type	Description
		'notifyonsave' user settings.
name_value_list.language	String	The language for the user.
name_value_list.notifyonsave	Boolean	Alerts users on new record creations when set to true.

Result

Name	Type	Description
result	entry_value | Array	The call result
result.id	String	This is the session id required to make other method calls.
result.module_name	String	Returns the 'Users' module.
result.name_value_list	Array	Authenticated user properties.
result.name_value_list.user_id	String	ID of the authenticated user.
result.name_value_list.user_name	String	Username of the authenticated user.
result.name_value_list.user_language	String	Default language of the authenticated user.
result.name_value_list.user_currency_id	String	Default currency ID of the authenticated user.
result.name_value_list.user_is_admin	String	Admin status of the authenticated user.
result.name_value_list.user_default_team_id	String	Default team of the authenticated user.
result.name_value_list.user_default_dateformat	String	Default date format for the authenticated user.
result.name_value_list.user_default_timeformat	String	Default time format for the authenticated user.
result.name_value_list.user_number_seperator	String	Number separator for the authenticated user.
result.name_value_list.use	String	Decimal separator for the

Name	Type	Description
r_decimal_seperator		authenticated user.
result.name_value_list.mobile_max_list_entries	String	Max list entires for the authenticated user.
result.name_value_list.mobile_max_subpanel_entries	String	Max subpanel entries for the authenticated user.
result.name_value_list.user_currency_name	String	Default currency name for the authenticated user.

Change Log

Version	Change
v3_1	Added additional return values to name_value_list. The list now also includes user_number_seperator, user_decimal_seperator, mobile_max_list_entries, mobile_max_subpanel_entries.
v3	Added additional return values to name_value_list. The list now also includes user_is_admin, user_default_team_id, user_default_dateformat, user_default_timeformat.
v2	Added name_value_list to response. Returns user_id, user_name, user_language, user_currency_id, user_currency_name.
v2	Added module_name to response.
v2	Removed error from response.
v2	Added name_value_list parameter
v2	Return type was changed from set_entry_result to entry_value.

PHP

```
$login_parameters = array(
    //user authentication
    "user_auth" => array(
        "user_name" => $username,
        "password" => md5($password),
    ),
```

```
    //application name
    "application_name" => "My Application",

    //name value list for 'language' and 'notifyonsave'
    "name_value_list" => array(
        array(
            'name' => 'language',
            'value' => 'en_us',
        ),

        array(
            'name' => 'notifyonsave',
            'value' => true
        ),
    ),
);
```

Last Modified: 09/26/2015 04:23pm

logout

Overview

Logs a user out of the Sugar application.

Available APIs

- SOAP
- REST

Definition

logout(session)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous call to login.

Result

Name	Type	Description
null	void	No return value.

Change Log

Version	Change
v2	Return type was changed from error_value to void.

PHP

```
$logout_parameters = array(
    //session id to expire
    "session" => $session_id,
);
```

Last Modified: 09/26/2015 04:23pm

oauth_access

Overview

Retrieves the OAuth access token.

Available APIs

- REST

Definition

oauth_access()

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.

Result

Name	Type	Description
result	Array	The call result.
result.id	String	The OAuth access token.

Change Log

Version	Change
v4	Added oauth_access method.

PHP

```
$oauth_access_parameters = array(
    //Session id
    'session' => $session_id,
);
```

Last Modified: 09/26/2015 04:23pm

seamless_login

Overview

Verifies that a session is authenticated.

Available APIs

- SOAP
- REST

Definition

seamless_login(session)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.

Result

Name	Type	Description
result	Integer	Returns 1 if the session is authenticated. Otherwise 0 will be returned.

Change Log

Version	Change

Considerations

If you are attempting to log a user into SugarCRM seamlessly, you can do this by passing the validated session_id in the url.

An example is shown below:

```
http://{site_url}/index.php?module=Home&action=index&MSID={session_id}
```

PHP

```
$seamless_login_parameters = array(
    //Session id
    "session" => $session_id,
);
```

Last Modified: 01/15/2016 10:05pm

search_by_module

Overview

Searches modules for a string and returns matched records.

Available APIs

- SOAP
- REST

Definition

search_by_module(session, search_string, modules, offset, max_results, assigned_user_id, select_fields, unified_search_only, favorites)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
search_string	String	The string to search for.
modules	Integer	The list of modules to query.
offset	Integer	The record offset from which to start.
max_results	Integer	The maximum number of records to return.
assigned_user_id	String	Filters records by the assigned user ID. Leave this empty if no filter should be applied.
select_fields	select_fields | Array	An array of fields to return. If empty the default return fields will be from the active listviewdefs.
unified_search_only	Boolean	If the search is only against modules participating in the unified search.
favorites	Boolean	If only records marked as favorites should be returned.

Result

Name	Type	Description
result	return_search_result | Array	Call result.
result.entry_list	Array	The count of records in paged result.
result.entry_list[].name	String	The .name of the module
result.entry_list[].records	Array	A list of name_value lists for each record matched.

Change Log

Version	Change
v3_1	Added unified_search_only parameter.

PHP

```
$search_by_module_parameters = array(
    //Session id
    "session" => $session_id,

    //The string to search for.
    'search_string' => 'example text',

    //The list of modules to query.
    'modules' => array(
      'Accounts',
    ),

    //The record offset from which to start.
    'offset' => 0,

    //The maximum number of records to return.
    'max_results' => 100,

    //Filters records by the assigned user ID.
    //Leave this empty if no filter should be applied.
    'assigned_user_id' => '',

    //An array of fields to return.
    //If empty the default return fields will be from the active
listviewdefs.
    'select_fields' => array(
      'id',
      'name',
    ),

    //If the search is only search modules participating in the
unified search.
    'unified_search_only' => false,

    //If only records marked as favorites should be returned.
```

```
    'favorites' => false
);
```

Last Modified: 09/26/2015 04:23pm

set_campaign_merge

Overview

Handles campaign log entry creation for mail-merge activity given a specified campaign.

Available APIs

- SOAP
- REST

Definition

set_campaign_merge(session, targets, campaign_id)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous call to login.
targets	select_fields | Array	A string array of IDs identifying the targets used in the merge. The IDs used in this parameter come from the column 'prospect_lists_prospects.id'.
campaign_id	String	The campaign ID used for the mail merge.

Result

Name	Type	Description
null	void	No return value.

Change Log

Version	Change
v2	Return type was changed from error_value to void.

PHP

```
$set_campaign_merge_parameters = array(
    //Session id
    "session" => $session_id,
    //A string array of IDs identifying the targets used in the merge.
    //The IDs used in this parameter come from the column
'prospect_lists_prospects.id'.
    "targets" => array(
        '403787cc-ab19-bec8-3ef4-50bd4896c9b3',
        'c5341c8d-4b0a-2b56-7108-50bd48b91213'
    ),
    //The campaign ID used for the mail merge.
    "campaign_id" => '781d4471-fb48-8dd2-ae62-50bd475950b2'
);
```

Last Modified: 09/26/2015 04:23pm

set_document_revision

Overview

Creates a new document revision for a specific document record.

Available APIs

- SOAP
- REST

Definition

set_document_revision(session, note)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
note	document_revision | Array	The file attachment details
note.id	String	The ID of the note record to associate the attachment to.
note.file	String	The binary contents of the file.
note.filename	String	The file name of the file attachment.
note.revision	String	The revision number

Result

Name	Type	Description
result	new_set_entry_result | array	The results from the call.
result.id	String	The ID of the document revision.

Change Log

Version	Change
v2	Return type was changed from set_entry_result to new_set_entry_result.

PHP

```
$file_contents = file_get_contents("/path/to/example_document.txt");
$set_document_revision_parameters = array(
    //Session id
    "session" => $session_id,

    //The attachment details
    "note" => array(
        //The ID of the parent document.
        'id' => $document_id,

        //The binary contents of the file.
        'file' => base64_encode($file_contents),

        //The name of the file
        'filename' => 'example_document.txt',

        //The revision number
        'revision' => '1',
    ),
);
```

Last Modified: 09/26/2015 04:23pm

set_entries

Overview

Create or update a list of records.

Available APIs

- SOAP
- REST

Definition

set_entries(session, module_name, name_value_lists)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
module_name	String	The name of the module from which to retrieve records. Note: This is the modules key which may not be the same as the modules display name.
name_value_lists	name_value_lists | Array	The an array of name/value lists containing the record attributes.

Result

Name	Type	Description
result	new_set_entries_result | Array	The call result.
result.ids	Array	The list of record IDs that were created or updated

Change Log

Version	Change
v2	Return type was changed from set_entry_result to new_set_entries_result.

Considerations

- To update an existing record, you will need to specify 'id' for the name_value_list item in the name_value_lists parameter.
- To create a new record with a specific ID, you will need to set 'new_with_id' in the name_value_list item in the name_value_lists parameter.

PHP

```
$set_entries_parameters = array(
    //Session id
    "session" => $session_id,

    //The name of the module from which to retrieve records.
    "module_name" => "Accounts",

    //Record attributes
    "name_value_lists" => array(
        array(
            //to update a record
            /*
            array(
                "name" => "id",
                "value" => "da0b107d-cfbc-cb08-4f90-50b7b9cb9ad7"
            ),
            */

            //to create a new record with a specific ID
            /*
            array(
                "name" => "new_with_id",
                "value" => 1
            ),
            */
            array(
                "name" => "name",
                "value" => "Example Account 1"
            ),
        ),

        array(
            //to update a record
            /*
            array(
```

```
                "name" => "id",
                "value" => "da0b107d-cfbc-cb08-4f90-50b7b9cb9ad7"
            ),
            */

            //to create a new record with a specific ID
            /*
            array(
                "name" => "new_with_id",
                "value" => 1
            ),
            */
            array(
                "name" => "name",
                "value" => "Example Account 2"
            ),
        ),
    ),
);
```

Last Modified: 09/26/2015 04:23pm

set_entry

Overview

Creates or updates a specific record.

Available APIs

- SOAP
- REST

Definition

set_entry(session, module_name, name_value_list)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
module_name	String	The name of the module from which to retrieve records. Note: This is the modules key which may not be the same as the modules display name.
name_value_list	name_value_list | Array	The name/value list of the record attributes.

Result

Name	Type	Description
result	new_set_entry_result | Array	The call result.
result.id	String	The ID of the record that was created/updated.

Change Log

Version	Change
v2	Return type was changed from set_entry_result to new_set_entry_result.

Considerations

- To update an existing record, you will need to specify 'id' in the name_value_list parameter.
- To create a new record with a specific ID, you will need to set 'new_with_id' in the name_value_list parameter.

PHP

```
$set_entry_parameters = array(
    //session id
    "session" => $session_id,

    //The name of the module from which to retrieve records.
    "module_name" => "Accounts",

    //Record attributes
    "name_value_list" => array(
        //to update a record
        /*
        array(
            "name" => "id",
            "value" => "da0b107d-cfbc-cb08-4f90-50b7b9cb9ad7"
        ),
        */

        //to create a new record with a specific ID
        /*
        array(
            "name" => "new_with_id",
            "value" => true
        ),
        */

        array(
            "name" => "name",
            "value" => "Example Account"
        ),
    ),
);
```

Last Modified: 09/26/2015 04:23pm

set_note_attachment

Overview

Creates an attachment and associated it to a specific note record.

Available APIs

- SOAP
- REST

Definition

set_note_attachment(session, note)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
note	new_note_attachment | Array	The file attachment details
note.id	String	The ID of the note record to associate the attachment to.
note.filename	String	The file name of the file attachment.
note.file	String	The binary contents of the file.

Result

Name	Type	Description
result	new_set_entry_result | Array	The call result.
result.id	String	The ID of the note record / attachment

Change Log

Version	Change

v2	Return type was changed from set_entry_result to new_set_entry_result.
v2	Parameter note type change from note_attachment to

PHP

```
$file_contents = file_get_contents("/path/to/example_file.php");
$set_note_attachment_parameters = array(
    //Session id
    "session" => $session_id,

    //The attachment details
    "note" => array(
        //The ID of the note containing the attachment.
        'id' => $note_id,

        //The file name of the attachment.
        'filename' => 'example_file.php',

        //The binary contents of the file.
        'file' => base64_encode($file_contents),
    ),
);
```

Last Modified: 09/26/2015 04:23pm

set_relationship

Overview

Sets relationships between two records. You can relate multiple records to a single record using this.

Available APIs

- SOAP
- REST

Definition

set_relationship(session, module_name, module_id, link_field_name, related_ids, name_value_list, delete)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
module_name	String	The name of the module from which to retrieve records. Note: This is the modules key which may not be the same as the modules display name.
module_id	String	The ID of the specified module record.
link_field_name	String	The name of the link field for the related module.
related_ids	select_fields | Array	The list of related record IDs you are relating
name_value_list	name_value_list | Array	An array specifying relationship fields to populate. An example of this is contact_role between Opportunities and Contacts.
delete	Integer	Determines whether the relationship is being created or deleted. 0:create, 1:delete

Result

Name	Type	Description
result	new_set_relationship_list_result | Array	The call result

Name	Type	Description
result.created	Integer	The number of relationships created.
result.failed	Integer	Determines whether or not the relationship failed. This is normally thrown when the parameters module_name or link_field_name are incorrect.
result.deleted	Integer	The number of relationships deleted.

Change Log

Version	Change
v2	Removed set_relationship_value parameter.
v2	Added module_name parameter.
v2	Added module_id parameter.
v2	Added link_field_name parameter.
v2	Added related_ids parameter.
v2	Added name_value_list parameter.
v2	Added delete parameter.
v2	Return type was changed from error_value to new_set_relationship_list_result.

PHP

```
$set_relationship_parameters = array(
    //session id
    'session' => $session_id,

    //The name of the module.
    'module_name' => 'Opportunities',

    //The ID of the specified module bean.
    'module_id' => '15e79b92-5025-827f-0784-50aa578270d8',
```

```
    //The relationship name of the linked field from which to relate
records.
    'link_field_name' => 'contacts',

    //The list of record ids to relate
    'related_ids' => array(
        '19b8799e-64ae-9502-588c-50aa575454c9',
    ),

    //Sets the value for relationship based fields
    'name_value_list' => array(
        array(
            'name' => 'contact_role',
            'value' => 'Other'
        ),
    ),

    //Whether or not to delete the relationship. 0:create, 1:delete
    'delete'=> 0,
);
```

Last Modified: 09/26/2015 04:23pm

set_relationships

Overview

Sets multiple relationships between multiple record sets.

Available APIs

- SOAP
- REST

Definition

set_relationships(session, module_names, module_ids, link_field_names, related_ids, name_value_lists, delete_array)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
module_names	select_fields | Array	The list of modules from which to retrieve records. Note: This is the modules key which may not be the same as the modules display name.
module_ids	select_fields | Array	The list of IDs for the specified module records.
link_field_names	select_fields | Array	The list of link names for the related modules.
related_ids	new_set_relationship_ids | Array	The list of related record IDs you are relating.
name_value_lists	name_value_lists | Array	An array of arrays specifying relationship fields to populate. An example of this is contact_role between Opportunities and Contacts.
delete_array	deleted_array | Array	An array determining whether the relationships are being created or deleted. 0:create, 1:delete

Result

Name	Type	Description
result	new_set_relationship_list_result | Array	The call result.
result.created	Integer	The number of relationships created.
result.failed	Integer	Determines whether or not the relationship failed.

Name	Type	Description
		This is normally thrown when the parameters module_name or link_field_name are incorrect.
result.deleted	Integer	The number of relationships deleted.

Change Log

Version	Change
v2	Removed set_relationship_value parameter.
v2	Added module_names parameter.
v2	Added module_ids parameter.
v2	Added link_field_names parameter.
v2	Added related_ids parameter.
v2	Added name_value_lists parameter.
v2	Added delete_array parameter.
v2	Return type was changed from set_relationship_list_result to new_set_relationship_list_result.

PHP

```
$set_relationships_parameters = array(
    //session id
    'session' => $session_id,

    //The name of the modules from which to relate records.
    'module_names' => array(
        'Opportunities',
        'Accounts',
    ),

    //The IDs of the specified module beans.
    'module_ids' => array(
        '15e79b92-5025-827f-0784-50aa578270d8', //Opportunity ID
```

```
        '27035f04-f6ec-492d-b89e-50aa57f5247f'  //Account ID
    ),

    //The relationship names of the linked fields from which to relate
records.
    'link_field_names' => array(
        'contacts', //Contacts link field to Opportunities
        'leads' //Leads link field to Accounts
    ),

    //The lists of record ids to relate
    'related_ids' => array(
        //Contact IDs
        array(
            '19b8799e-64ae-9502-588c-50aa575454c9'
        ),

        //Lead IDs
        array(
            '16d8d519-5f56-0984-2092-50aa576a7333',
            '15ae07eb-63f0-dbac-6e4c-50aa57c5a609'
        ),
    ),

    //Sets the value for relationship based fields
    'name_value_lists' => array(
        //Opportunity-Contact relationship fields
        array(
            array(
                'name' => 'contact_role',
                'value' => 'Other'
            ),
        ),

        //Account-Lead relationship fields
        array(
        ),
    ),

    //Whether or not to delete the relationships. 0:create, 1:delete
    'delete_array'=> array(
        0, //Opportunity-Contact
        0  //Account-Lead
    ),
);
```

Last Modified: 09/26/2015 04:23pm

snip_import_emails

Overview

Used to imports an email record from the SNIP archiving service.

Available APIs

- REST

Definition

snip_import_emails(session, email)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
email	Array	The contents of the email being imported.
email.message	Array	Contains the email attributes.
email.message.message_id	String	The ID of the email message.
email.message.subject	String	Email subject.
email.message.attachments	Array	The list of attachments to be imported
email.message.from_name	String	From sender name.
email.message.description	String	Plain text content body.
email.message.description_html	String	HTML email content body.

Name	Type	Description
email.message.to_addrs	String	Email addresses the email was sent to.
email.message.cc_addrs	String	Email addresses the email was CCed to.
email.message.bcc_addrs	String	Email addresses the email was BCCed to.
email.message.date_sent	String	Date the email was sent.

Result

Name	Type	Description
result	Array	The call result.
result.results	Boolean	The success of the import.
result.count	Integer	The count of records imported.
result.message	String	The return message.

Change Log

Version	Change
v4	Added snip_import_emails method.

Last Modified: 09/26/2015 04:23pm

snip_update_contacts

Overview

Retrieves new contact emails since a timestamp for the current user.

Available APIs

- REST

Definition

snip_update_contacts(session, report_id)

Parameters

Name	Type	Description
session	String	Session ID returned by a previous login call.
report_id	String	The ID of the report record to generate the PDF for.

Result

Name	Type	Description
result	Array	The call result.
result.results	Boolean	The new emails.
result.count	Integer	The count of results.
result.message	String	The return message.

Change Log

Version	Change
v4	Added snip_update_contacts method.

Last Modified: 09/26/2015 04:23pm

Extending Web Services

Overview

The guide will demonstrate how to add your own custom methods to the REST and SOAP API or extend exisiting ones.

Extending the API

The following example will demonstrate how to extend the v4_1 API.

Defining the Entry Point Location

This is where you define the directory that will contain your new REST and SOAP entry points. We recommend a path formatted as follows:

```
./custom/service/{version}_custom/
```

The actual location of the entry points does not matter, however, using a path such as this will allow you to call your entry points as follows:

- http://{sugar_url}/custom/service/{version}_custom/rest.php
- http://{sugar_url}/custom/service/{version}_custom/soap.php

Define the SugarWebServiceImpl Class

The next step is to define a new 'SugarWebServiceImpl' class. Since we are using v4_1, we need to extend the file 'service/v4_1/SugarWebServiceImplv4_1.php' and add our new method. To do this, we will create the file:

./custom/service/v4_1_custom/SugarWebServiceImplv4_1_custom.php

```
<?php

    if(!defined('sugarEntry'))define('sugarEntry', true);
    require_once('service/v4_1/SugarWebServiceImplv4_1.php');

    class SugarWebServiceImplv4_1_custom extends
SugarWebServiceImplv4_1
    {
        /*
         * Returns the session id if authenticated
         *
```

```
         * @param string $session
         * @return string $session - false if invalid.
         *
         */

        function example_method($session)
        {
            $GLOBALS['log']->info('Begin:
SugarWebServiceImplv4_1_custom->example_method');
            $error = new SoapError();

            //authenticate
            if
(!self::$helperObject->checkSessionAndModuleAccess($session,
'invalid_session', '', '', '',  $error))
            {
                $GLOBALS['log']->info('End:
SugarWebServiceImplv4_1_custom->example_method.');
                return false;
            }

            return $session;
        }
    }

?>
```

Define the Registry Class

Next, we will define the registry class that will register our new function. This file will be located at:

./custom/service/v4_1_custom/registry.php

```
<?php

    require_once('service/v4_1/registry.php');

    class registry_v4_1_custom extends registry_v4_1
    {
        protected function registerFunction()
        {
            parent::registerFunction();
            $this->serviceClass->registerFunction('example_method',
array('session'=>'xsd:string'), array('return'=>'xsd:string'));
```

```
        }
    }

?>
```

Define the REST Entry Point

This REST entry point will be located at:

./custom/service/v4_1_custom/rest.php

```
<?php

    chdir('../../..');

    require_once('SugarWebServiceImplv4_1_custom.php');

    $webservice_path = 'service/core/SugarRestService.php';
    $webservice_class = 'SugarRestService';
    $webservice_impl_class = 'SugarWebServiceImplv4_1_custom';

    $registry_path = 'custom/service/v4_1_custom/registry.php';
    $registry_class = 'registry_v4_1_custom';
    $location = 'custom/service/v4_1_custom/rest.php';
    require_once('service/core/webservice.php');

?>
```

Define the SOAP Entry Point

This SOAP entry point will be located at:

./custom/service/v4_1_custom/soap.php

```
<?php

    chdir('../../..');

    require_once('SugarWebServiceImplv4_1_custom.php');

    $webservice_class = 'SugarSoapService2';
    $webservice_path = 'service/v2/SugarSoapService2.php';
    $webservice_impl_class = 'SugarWebServiceImplv4_1_custom';
```

```
    $registry_class = 'registry_v4_1_custom';
    $registry_path = 'custom/service/v4_1_custom/registry.php';
    $location = 'custom/service/v4_1_custom/soap.php';

    require_once('service/core/webservice.php');
?>
```

Last Modified: 09/26/2015 04:23pm

REST Release Notes

Overview

Lists changes between the different versions of the REST API.

Release Notes

v4_1

- get_modified_relationships method was added.
- get_relationships had the parameter $limit added.
- get_relationships had the parameter $offset added.

v4

- get_entries had the parameter $track_view removed.
- job_queue_cycle method was added.
- job_queue_next method was added.
- job_queue_run method was added.
- oauth_access method was added.
- oauth_access_token method was added.
- oauth_request_token method was added.
- search_by_module had the parameter $favorites added.
- snip_import_emails method was added.
- snip_update_contacts method was added.

v3_1

- get_entries had the parameter $track_view added.

- get_entry had the parameter $track_view added.
- get_entry_list had the parameter $favorites added.
- get_language_definition method was added.
- get_module_layout had the parameter $acl_check added.
- get_module_layout_md5 had the parameter $acl_check added.
- get_quotes_pdf method was added.
- get_report_pdf method was added.
- search_by_module had the parameter $unified_search_only added.
- set_entry had the parameter $track_view added.

v3

- get_available_modules had the parameter $filter added.
- get_last_viewed method was added.
- get_module_fields_md5 method was added.
- get_module_layout method was added.
- get_module_layout_md5 method was added.
- get_relationships had the parameter $order_by added.
- get_upcoming_activities method was added.
- search_by_module had the parameter $assigned_user_id added.
- search_by_module had the parameter $select_fields added.

v2_1

- get_entry_list method logic was modified.
- get_report_entries method logic was modified.
- md5 method was removed.

v2 (REST API was introduced into SugarCRM)

- get_available_modules method was added.
- get_document_revision method was added.
- get_entries method was added.
- get_entries_count method was added.
- get_entry method was added.
- get_entry_list method was added.
- get_module_fields method was added.
- get_note_attachment method was added.
- get_relationships method was added.
- get_report_entries method was added.
- get_server_info method was added.

- get_user_id method was added.
- get_user_team_id method was added.
- login method was added.
- logout method was added.
- md5 method was added.
- seamless_login method was added.
- search_by_module method was added.
- set_campaign_merge method was added.
- set_document_revision method was added.
- set_entries method was added.
- set_entry method was added.
- set_note_attachment method was added.
- set_relationship method was added.
- set_relationships method was added.SOAP Release Notes

Last Modified: 11/19/2015 03:52am

SOAP Release Notes

Overview

Lists changes between the different versions of the SOAP API.

Release Notes

v4_1

- get_modified_relationships method was added.
- get_relationships had the parameter $limit added.
- get_relationships had the parameter $offset added.

v4

- search_by_module had the parameter $favorites added.

v3_1

- get_entries had the parameter $track_view added.
- get_entry had the parameter $track_view added.

- get_entry_list had the parameter $favorites added.
- search_by_module had the parameter $unified_search_only added.

v3

- get_available_modules had the parameter $filter added.
- get_last_viewed method was added.
- get_module_fields_md5 method was added.
- get_relationships had the parameter $order_by added.
- get_upcoming_activities method was added.
- search_by_module had the parameter $assigned_user_id added.
- search_by_module had the parameter $select_fields added.

v2_1

- get_entry_list method logic was modified.
- get_report_entries method logic was modified.

v2

- contact_by_email method was removed.
- create_account method was removed.
- create_case method was removed.
- create_contact method was removed.
- create_lead method was removed.
- create_opportunity method was removed.
- create_session method was removed.
- end_session method was removed.
- get_attendee_list method was removed.
- get_contact_relationships method was removed.
- get_disc_client_file_list method was removed.
- get_document_revision return type was changed from return_document_revision to new_return_document_revision.
- get_encoded_file method was removed.
- get_encoded_portal_zip_file method was removed.
- get_encoded_zip_file method was removed.
- get_entries had the parameter $link_name_to_fields_array added.
- get_entries return type was changed from get_entry_result to get_entry_result_version2.
- get_entry had the parameter $link_name_to_fields_array added.
- get_entry return type was changed from get_entry_result to

get_entry_result_version2.
- get_entry_list had the parameter $link_name_to_fields_array added.
- get_entry_list return type was changed from get_entry_list_result to get_entry_list_result_version2.
- get_gmt_time method was removed.
- get_mailmerge_document method was removed.
- get_mailmerge_document2 method was removed.
- get_modified_entries method was removed.
- get_module_fields had the parameter $fields added.
- get_module_fields return type was changed from module_fields to new_module_fields.
- get_note_attachment return type was changed from return_note_attachment to new_return_note_attachment.
- get_quick_sync_data method was removed.
- get_related_notes method was removed.
- get_relationships had the parameter $link_field_name added.
- get_relationships had the parameter $related_fields added.
- get_relationships had the parameter $related_module removed.
- get_relationships had the parameter $related_module_link_name_to_fields_array added.
- get_relationships return type was changed from get_relationships_result to get_entry_result_version2.
- get_report_entries method was added.
- get_required_upgrades method was removed.
- get_server_info method was added.
- get_server_time method was removed.
- get_server_version method was removed.
- get_sugar_flavor method was removed.
- get_system_status method was removed.
- get_unique_system_id method was removed.
- is_loopback method was removed.
- is_user_admin method was removed.
- login had the parameter $name_value_list added.
- login return type was changed from set_entry_result to entry_value.
- logout return type was changed from error_value to void.
- offline_client_available method was removed.
- relate_note_to_module method was removed.
- search method was removed.
- search_by_module had the parameter $password removed.
- search_by_module had the parameter $session added.
- search_by_module had the parameter $user_name removed.
- search_by_module return type was changed from get_entry_list_result to return_search_result.
- set_campaign_merge return type was changed from error_value to void.
- set_document_revision return type was changed from set_entry_result to new_set_entry_result.

- set_entries return type was changed from set_entries_result to new_set_entries_result.
- set_entries_details method was removed.
- set_entry return type was changed from set_entry_result to new_set_entry_result.
- set_note_attachment had the parameter $note type change from note_attachment to new_note_attachment.
- set_note_attachment return type was changed from set_entry_result to new_set_entry_result.
- set_relationship had the parameter $delete added.
- set_relationship had the parameter $link_field_name added.
- set_relationship had the parameter $module_id added.
- set_relationship had the parameter $module_name added.
- set_relationship had the parameter $name_value_list added.
- set_relationship had the parameter $related_ids added.
- set_relationship had the parameter $set_relationship_value removed.
- set_relationship return type was changed from error_value to new_set_relationship_list_result.
- set_relationships had the parameter $delete_array added.
- set_relationships had the parameter $link_field_names added.
- set_relationships had the parameter $module_ids added.
- set_relationships had the parameter $module_names added.
- set_relationships had the parameter $name_value_lists added.
- set_relationships had the parameter $related_ids added.
- set_relationships had the parameter $set_relationship_list removed.
- set_relationships return type was changed from set_relationship_list_result to new_set_relationship_list_result.
- sudo_user method was removed.
- sync_get_entries method was removed.
- sync_get_modified_relationships method was removed.
- sync_get_relationships method was removed.
- sync_set_entries method was removed.
- sync_set_relationships method was removed.
- track_email method was removed.
- update_portal_user method was removed.
- user_list method was removed.

Last Modified: 11/19/2015 03:52am

Examples

Examples of v4_1 Web Service API Calls.

Last Modified: 11/19/2015 03:52am

REST

Examples of v4_1 REST API Calls.

Last Modified: 11/19/2015 03:52am

PHP

PHP v4_1 REST Examples.

Last Modified: 11/19/2015 04:22am

Creating Documents

Overview

A PHP example demonstrating how to create a document using set_entry and a document revision with the set_document_revision method using cURL and the v4_1 REST API.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/rest.php";

    $username = "admin";
    $password = "password";

    //function to make cURL request
    function call($method, $parameters, $url)
    {
        ob_start();
        $curl_request = curl_init();

        curl_setopt($curl_request, CURLOPT_URL, $url);
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
```

```
    curl_setopt($curl_request, CURLOPT_HEADER, 1);
    curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
    curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

    $jsonEncodedData = json_encode($parameters);

    $post = array(
         "method" => $method,
         "input_type" => "JSON",
         "response_type" => "JSON",
         "rest_data" => $jsonEncodedData
    );

    curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);
    $result = curl_exec($curl_request);
    curl_close($curl_request);

    $result = explode("\r\n\r\n", $result, 2);
    $response = json_decode($result[1]);
    ob_end_flush();

    return $response;
}

//login ----------------------------------------------------
$login_parameters = array(
     "user_auth" => array(
          "user_name" => $username,
          "password" => md5($password),
          "version" => "1"
     ),
     "application_name" => "RestTest",
     "name_value_list" => array(),
);

$login_result = call("login", $login_parameters, $url);

/*
echo "<pre>";
print_r($login_result);
echo "</pre>";
*/

//get session id
$session_id = $login_result->id;
```

```
    //create document ---------------------------------------------
    $set_entry_parameters = array(
        //session id
        "session" => $session_id,

        //The name of the module
        "module_name" => "Documents",

        //Record attributes
        "name_value_list" => array(
            //to update a record, pass in a record id as commented
below
            //array("name" => "id", "value" =>
"9b170af9-3080-e22b-fbc1-4fea74def88f"),
            array("name" => "document_name", "value" => "Example
Document"),
            array("name" => "revision", "value" => "1"),
        ),
    );

    $set_entry_result = call("set_entry", $set_entry_parameters,
$url);

    echo "<pre>";
    print_r($set_entry_result);
    echo "</pre>";

    $document_id = $set_entry_result->id;

    //create document revision ------------------------------------
    $contents = file_get_contents ("/path/to/example_document.txt");

    $set_document_revision_parameters = array(
        //session id
        "session" => $session_id,

        //The attachment details
        "note" => array(
            //The ID of the parent document.
            'id' => $document_id,

            //The binary contents of the file.
            'file' => base64_encode($contents),

            //The name of the file
```

```
            'filename' => 'example_document.txt',

            //The revision number
            'revision' => '1',
        ),
    );

    $set_document_revision_result = call("set_document_revision",
$set_document_revision_parameters, $url);

    echo "<pre>";
    print_r($set_document_revision_result);
    echo "</pre>";

?>
```

Result

```
//set_entry result
stdClass Object
(
    [id] => b769cf46-7881-a369-314d-50abaa238c62
    [entry_list] => stdClass Object
        (
            [document_name] => stdClass Object
                (
                    [name] => document_name
                    [value] => Example Document
                )

            [revision] => stdClass Object
                (
                    [name] => revision
                    [value] => 1
                )
        )
)

//set_document_revision result
stdClass Object
(
    [id] => e83f97b9-b818-2d04-1aeb-50abaa8303b5
)
```

Last Modified: 09/26/2015 04:23pm

Creating Notes with Attachments

Overview

A PHP example demonstrating how to create a note using set_entry and add an attachment with the set_note_attachment method using cURL and the v4_1 REST API.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/rest.php";
    $username = "admin";
    $password = "password";

    //function to make cURL request
    function call($method, $parameters, $url)
    {
        ob_start();
        $curl_request = curl_init();

        curl_setopt($curl_request, CURLOPT_URL, $url);
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
        curl_setopt($curl_request, CURLOPT_HEADER, 1);
        curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

        $jsonEncodedData = json_encode($parameters);

        $post = array(
             "method" => $method,
             "input_type" => "JSON",
             "response_type" => "JSON",
             "rest_data" => $jsonEncodedData
        );

        curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);
        $result = curl_exec($curl_request);
```

```
        curl_close($curl_request);

        $result = explode("\r\n\r\n", $result, 2);
        $response = json_decode($result[1]);
        ob_end_flush();

        return $response;
    }

    //login -------------------------------------------------
    $login_parameters = array(
         "user_auth" => array(
              "user_name" => $username,
              "password" => md5($password),
              "version" => "1"
         ),
         "application_name" => "RestTest",
         "name_value_list" => array(),
    );

    $login_result = call("login", $login_parameters, $url);

    /*
    echo "<pre>";
    print_r($login_result);
    echo "</pre>";
    */

    //get session id
    $session_id = $login_result->id;

    //create note -------------------------------------------
    $set_entry_parameters = array(
         //session id
         "session" => $session_id,

         //The name of the module
         "module_name" => "Notes",

         //Record attributes
         "name_value_list" => array(
              //to update a record, you will nee to pass in a record
id as commented below
              //array("name" => "id", "value" =>
"9b170af9-3080-e22b-fbc1-4fea74def88f"),
              array("name" => "name", "value" => "Example Note"),
```

```
        ),
    );

    $set_entry_result = call("set_entry", $set_entry_parameters,
$url);

    echo "<pre>";
    print_r($set_entry_result);
    echo "</pre>";

    $note_id = $set_entry_result->id;

    //create note attachment ----------------------------------
    $contents = file_get_contents ("/path/to/example_file.php");

    $set_note_attachment_parameters = array(
        //session id
        "session" => $session_id,

        //The attachment details
        "note" => array(
            //The ID of the note containing the attachment.
            'id' => $note_id,

            //The file name of the attachment.
            'filename' => 'example_file.php',

            //The binary contents of the file.
            'file' => base64_encode($contents),
        ),
    );

    $set_note_attachment_result = call("set_note_attachment",
$set_note_attachment_parameters, $url);

    echo "<pre>";
    print_r($set_note_attachment_result);
    echo "</pre>";

?>
```

Result

```
//set_entry result
stdClass Object
```

```
(
    [id] => 72508938-db19-3b5c-b7a8-50abc7ec3fdb
    [entry_list] => stdClass Object
        (
            [name] => stdClass Object
                (
                    [name] => name
                    [value] => Example Note
                )
        )
)

//set_note_attachment result
stdClass Object
(
    [id] => 72508938-db19-3b5c-b7a8-50abc7ec3fdb
)
```

Last Modified: 09/26/2015 04:23pm

Creating or Updating a Record

Overview

A PHP example demonstrating how to create or update an Account with the set_entry method using cURL and the v4_1 REST API.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/rest.php";
    $username = "admin";
    $password = "password";

    //function to make cURL request
    function call($method, $parameters, $url)
    {
        ob_start();
        $curl_request = curl_init();

        curl_setopt($curl_request, CURLOPT_URL, $url);
```

```
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
        curl_setopt($curl_request, CURLOPT_HEADER, 1);
        curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

        $jsonEncodedData = json_encode($parameters);

        $post = array(
             "method" => $method,
             "input_type" => "JSON",
             "response_type" => "JSON",
             "rest_data" => $jsonEncodedData
        );

        curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);
        $result = curl_exec($curl_request);
        curl_close($curl_request);

        $result = explode("\r\n\r\n", $result, 2);
        $response = json_decode($result[1]);
        ob_end_flush();

        return $response;
    }

    //login ------------------------------------------
    $login_parameters = array(
         "user_auth" => array(
              "user_name" => $username,
              "password" => md5($password),
              "version" => "1"
         ),
         "application_name" => "RestTest",
         "name_value_list" => array(),
    );

    $login_result = call("login", $login_parameters, $url);

    /*
    echo "<pre>";
    print_r($login_result);
    echo "</pre>";
    */
```

```
    //get session id
    $session_id = $login_result->id;

    //create account -------------------------------------
    $set_entry_parameters = array(
         //session id
         "session" => $session_id,

         //The name of the module from which to retrieve records.
         "module_name" => "Accounts",

         //Record attributes
         "name_value_list" => array(
              //to update a record, you will nee to pass in a record
id as commented below
              //array("name" => "id", "value" =>
"9b170af9-3080-e22b-fbc1-4fea74def88f"),
              array("name" => "name", "value" => "Test Account"),
         ),
    );

    $set_entry_result = call("set_entry", $set_entry_parameters,
$url);

    echo "<pre>";
    print_r($set_entry_result);
    echo "</pre>";

?>
```

Result

```
stdClass Object
(
    [id] => 9b170af9-3080-e22b-fbc1-4fea74def88f
    [entry_list] => stdClass Object
        (
            [name] => stdClass Object
                (
                    [name] => name
                    [value] => Test Account
                )
        )
)
```

Last Modified: 09/26/2015 04:23pm

Creating or Updating Multiple Records

Overview

A PHP example demonstrating how to create or update multiple contacts with the set_entries method using cURL and the v4_1 REST API.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/rest.php";
    $username = "admin";
    $password = "password";

    //function to make cURL request
    function call($method, $parameters, $url)
    {
        ob_start();
        $curl_request = curl_init();

        curl_setopt($curl_request, CURLOPT_URL, $url);
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
        curl_setopt($curl_request, CURLOPT_HEADER, 1);
        curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

        $jsonEncodedData = json_encode($parameters);

        $post = array(
             "method" => $method,
             "input_type" => "JSON",
             "response_type" => "JSON",
             "rest_data" => $jsonEncodedData
        );
```

```
        curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);
        $result = curl_exec($curl_request);
        curl_close($curl_request);

        $result = explode("\r\n\r\n", $result, 2);
        $response = json_decode($result[1]);
        ob_end_flush();

        return $response;
    }

    //login ------------------------------------------
    $login_parameters = array(
         "user_auth" => array(
              "user_name" => $username,
              "password" => md5($password),
              "version" => "1"
         ),
         "application_name" => "RestTest",
         "name_value_list" => array(),
    );

    $login_result = call("login", $login_parameters, $url);

    /*
    echo "<pre>";
    print_r($login_result);
    echo "</pre>";
    */

    //get session id
    $session_id = $login_result->id;

    //create contacts ----------------------------------
    $set_entries_parameters = array(
         //session id
         "session" => $session_id,

         //The name of the module from which to retrieve records.
         "module_name" => "Contacts",

         //Record attributes
         "name_value_list" => array(
             array(
                //to update a record, you will nee to pass in a record
id as commented below
```

```
                //array("name" => "id", "value" =>
"912e58c0-73e9-9cb6-c84e-4ff34d62620e"),
                array("name" => "first_name", "value" => "John"),
                array("name" => "last_name", "value" => "Smith"),
            ),
            array(
                //to update a record, you will nee to pass in a record
id as commented below
                //array("name" => "id", "value" =>
"99d6ddfd-7d52-d45b-eba8-4ff34d684964"),
                array("name" => "first_name", "value" => "Jane"),
                array("name" => "last_name", "value" => "Doe"),
            ),
        ),
    );

    $set_entries_result = call("set_entries", $set_entries_parameters,
$url);

    echo "<pre>";
    print_r($set_entries_result);
    echo "</pre>";

?>
```

Result

```
stdClass Object
(
    [ids] => Array
        (
            [0] => 912e58c0-73e9-9cb6-c84e-4ff34d62620e
            [1] => 99d6ddfd-7d52-d45b-eba8-4ff34d684964
        )
)
```

Last Modified: 09/26/2015 04:23pm

Creating or Updating Teams

Overview

A PHP example demonstrating how to manipulate teams using cURL and the v4_1 REST API.

Note: If you are creating a private team for a user, you will need to set private to true and populate the associated_user_id populated. You should also populate the name and name_2 properties with the users first and last name.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/rest.php";
    $username = "admin";
    $password = "password";

    //function to make cURL request
    function call($method, $parameters, $url)
    {
        ob_start();
        $curl_request = curl_init();

        curl_setopt($curl_request, CURLOPT_URL, $url);
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
        curl_setopt($curl_request, CURLOPT_HEADER, 1);
        curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

        $jsonEncodedData = json_encode($parameters);

        $post = array(
            "method" => $method,
            "input_type" => "JSON",
            "response_type" => "JSON",
            "rest_data" => $jsonEncodedData
        );

        curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);
        $result = curl_exec($curl_request);
        curl_close($curl_request);

        $result = explode("\r\n\r\n", $result, 2);
        $response = json_decode($result[1]);
```

```
        ob_end_flush();

        return $response;
    }

    //login ------------------------------------------
    $login_parameters = array(
        "user_auth" => array(
            "user_name" => $username,
            "password" => md5($password),
            "version" => "1"
        ),
        "application_name" => "RestTest",
        "name_value_list" => array(),
    );

    $login_result = call("login", $login_parameters, $url);

    /*
    echo "<pre>";
    print_r($login_result);
    echo "</pre>";
    */

    //get session id
    $session_id = $login_result->id;

    //create team ------------------------------------------------
    $set_entry_parameters = array(

        // session id
        "session" => $session_id,

        // The name of the module that the record will be create in.
        "module_name" => "Teams",

        // array of arrays for the record attributes
        "name_value_list" => array(
            /* Setting the id with a valid record id will update the
record.
            array(
                "name" => "id",
                "value" => "47dbab1d-bd78-09e8-4392-5256b4501d90"
            ),
            */
            array(
```

```
                "name" => "name",
                "value" => "My Team"
            ),
            array(
                "name" => "description",
                "value" => "My new team"
            ),
            //Whether the team is private.
            //Private teams will have the associated_user_id
populated.
            array(
                "name" => "private",
                "value" => 0
            ),
        ),
    );

    $set_entry_result = call("set_entry", $set_entry_parameters,
$url);

    echo "<pre>";
    print_r($set_entry_result);
    echo "</pre>";

?>
```

Result

```
stdClass Object
(
    [id] => 5c35a3be-4601-fb45-3afd-52ab78b03f89
    [entry_list] => stdClass Object
        (
            [name] => stdClass Object
                (
                    [name] => name
                    [value] => My Team
                )

            [description] => stdClass Object
                (
                    [name] => description
                    [value] => My new team
                )
```

```
            [private] => stdClass Object
                (
                    [name] => private
                    [value] =>
                )
        )
)
```

Last Modified: 09/26/2015 04:23pm

Logging In

Overview

A PHP example demonstrating how to log in and retrieve a session key using cURL and the v4_1 REST API.

Standard Authentication Example

```
<?php

    $url = "http://{site_url}/service/v4_1/rest.php";
    $username = "admin";
    $password = "password";

    //function to make cURL request
    function call($method, $parameters, $url)
    {
        ob_start();
        $curl_request = curl_init();

        curl_setopt($curl_request, CURLOPT_URL, $url);
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
        curl_setopt($curl_request, CURLOPT_HEADER, 1);
        curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

        $jsonEncodedData = json_encode($parameters);
```

```
        $post = array(
             "method" => $method,
             "input_type" => "JSON",
             "response_type" => "JSON",
             "rest_data" => $jsonEncodedData
        );

        curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);
        $result = curl_exec($curl_request);
        curl_close($curl_request);

        $result = explode("\r\n\r\n", $result, 2);
        $response = json_decode($result[1]);
        ob_end_flush();

        return $response;
    }

    //login ----------------------------
    $login_parameters = array(
         "user_auth" => array(
              "user_name" => $username,
              "password" => md5($password),
              "version" => "1"
         ),
         "application_name" => "RestTest",
         "name_value_list" => array(),
    );

    $login_result = call("login", $login_parameters, $url);

    echo "<pre>";
    print_r($login_result);
    echo "</pre>";

    //get session id
    $session_id = $login_result->id;

?>
```

LDAP Authentication Example

```
<?php
```

```
    $url = "http://{site_url}/service/v4_1/rest.php";
    $username = "admin";
    $password = "password";
    $ldap_enc_key = 'LDAP_ENCRYPTION_KEY';

    //function to make cURL request
    function call($method, $parameters, $url)
    {
        ob_start();

        $curl_request = curl_init();
        curl_setopt($curl_request, CURLOPT_URL, $url);
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
        curl_setopt($curl_request, CURLOPT_HEADER, 1);
        curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

        $jsonEncodedData = json_encode($parameters);

        $post = array(
             "method" => $method,
             "input_type" => "JSON",
             "response_type" => "JSON",
             "rest_data" => $jsonEncodedData
        );

        curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);

        $result = curl_exec($curl_request);

        curl_close($curl_request);

        $result = explode("\r\n\r\n", $result, 2);
        $response = json_decode($result[1]);

        ob_end_flush();

        return $response;
    }

    //login ------------------------------
    $ldap_enc_key = substr(md5($ldap_enc_key), 0, 24);
    $login_parameters = array(
```

```
        "user_auth" => array(
             "user_name" => $username,
             "password" => bin2hex(mcrypt_cbc(MCRYPT_3DES,
$ldap_enc_key, $password, MCRYPT_ENCRYPT, 'password')),
             "version" => "1"
        ),

        "application_name" => "RestTest",
        "name_value_list" => array(),
    );

    $login_result = call("login", $login_parameters, $url);

    echo "<pre>";
    print_r($login_result);
    echo "</pre>";

    //get session id
    $session_id = $login_result->id;

?>
```

Result

```
stdClass Object
(
    [id] => lb7479svj8pjtf57ipmshepo80
    [module_name] => Users
    [name_value_list] => stdClass Object
        (
            [user_id] => stdClass Object
                (
                    [name] => user_id
                    [value] => 1
                )

            [user_name] => stdClass Object
                (
                    [name] => user_name
                    [value] => admin
                )

            [user_language] => stdClass Object
                (
                    [name] => user_language
```

```
            [value] => en_us
        )

    [user_currency_id] => stdClass Object
        (
            [name] => user_currency_id
            [value] => -99
        )

    [user_is_admin] => stdClass Object
        (
            [name] => user_is_admin
            [value] => 1
        )

    [user_default_team_id] => stdClass Object
        (
            [name] => user_default_team_id
            [value] => 1
        )

    [user_default_dateformat] => stdClass Object
        (
            [name] => user_default_dateformat
            [value] => m/d/Y
        )

    [user_default_timeformat] => stdClass Object
        (
            [name] => user_default_timeformat
            [value] => h:ia
        )

    [user_number_seperator] => stdClass Object
        (
            [name] => user_number_seperator
            [value] => ,
        )

    [user_decimal_seperator] => stdClass Object
        (
            [name] => user_decimal_seperator
            [value] => .
        )

    [mobile_max_list_entries] => stdClass Object
```

```
                (
                    [name] => mobile_max_list_entries
                    [value] => 10
                )

            [mobile_max_subpanel_entries] => stdClass Object
                (
                    [name] => mobile_max_subpanel_entries
                    [value] => 3
                )

            [user_currency_name] => stdClass Object
                (
                    [name] => user_currency_name
                    [value] => US Dollars
                )
        )
)
```

Last Modified: 09/26/2015 04:23pm

Relating Quotes and Products

Overview

A PHP example demonstrating how to create and relate Products to Quotes using cURL and the v4_1 REST API.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/rest.php";
    $username = "admin";
    $password = "password";

    //function to make cURL request
    function call($method, $parameters, $url)
    {
        ob_start();
        $curl_request = curl_init();
```

```
        curl_setopt($curl_request, CURLOPT_URL, $url);
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
        curl_setopt($curl_request, CURLOPT_HEADER, 1);
        curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

        $jsonEncodedData = json_encode($parameters);

        $post = array(
            "method" => $method,
            "input_type" => "JSON",
            "response_type" => "JSON",
            "rest_data" => $jsonEncodedData
        );

        curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);
        $result = curl_exec($curl_request);
        curl_close($curl_request);

        $result = explode("\r\n\r\n", $result, 2);
        $response = json_decode($result[1]);
        ob_end_flush();

        return $response;
    }

    //login ------------------------------------------
    $login_parameters = array(
        "user_auth" => array(
            "user_name" => $username,
            "password" => md5($password),
            "version" => "1"
        ),
        "application_name" => "RestTest",
        "name_value_list" => array(),
    );

    $login_result = call("login", $login_parameters, $url);

    /*
    echo "<pre>";
    print_r($login_result);
    echo "</pre>";
```

```
    */

    //get session id
    $session_id = $login_result->id;

    //create quote -----------------------------------------------
    $createQuoteParams = array(
        'session' => $session_id,
        'module_name' => 'Quotes',
        'name_value_list' => array(
            array(
                'name' => 'name',
                'value' => 'Widget Quote'
            ),
            array(
                'name' => 'team_count',
                'value' => ''
            ),
            array(
                'name' => 'team_name',
                'value' => ''
            ),
            array(
                'name' => 'date_quote_expected_closed',
                'value' => date('Y-m-d', mktime(0, 0, 0, date('m')  ,
date('d')+7, date('Y')))
            ),
            array(
                'name' => 'quote_stage',
                'value' => 'Negotiation'
            ),
            array(
                'name' => 'quote_num',
                'value' => ''
            ),
            array(
                'name' => 'quote_type',
                'value' => 'Quotes'
            ),
            array(
                'name' => 'subtotal',
                'value' => '1230.23'
            ),
            array(
                'name' => 'subtotal_usdollar',
                'value' => '1230.23'
```

```
            ),
        ),
    );

    $createQuoteResult = call('set_entry', $createQuoteParams, $url);

    echo "Create Quote Result<br />";
    echo "<pre>";
    print_r($createQuoteResult);
    echo "</pre>";

    //create product ---------------------------------------------
    $createProductParams = array(
        'session' => $session_id,
        'module_name' => 'Products',
        'name_value_list' => array(
            array(
                'name' => 'name',
                'value' => 'Widget'
            ),
            array(
                'name' => 'quote_id',
                'value' => $createQuoteResult->id
            ),
            array(
                'name' => 'status',
                'value' => 'Quotes'
            )
        )
    );

    $createProductResult = call('set_entry', $createProductParams,
$url);

    echo "Create Product Result<br />";
    echo "<pre>";
    print_r($createProductResult);
    echo "</pre>";

    //create product-bundle
-------------------------------------------
    $createProductBundleParams = array(
        "session"         => $session_id,
        "module_name"     => "ProductBundles",
        "name_value_list" => array(
            array(
```

```
                'name' => 'name',
                'value' => 'Rest SugarOnline Order'),
            array(
                'name' => 'bundle_stage',
                'value' => 'Draft'
            ),
            array(
                'name' => 'tax',
                'value' => '0.00'
            ),
            array(
                'name' => 'total',
                'value' => '0.00'
            ),
            array(
                'name' => 'subtotal',
                'value' => '0.00'
            ),
            array(
                'name' => 'shipping',
                'value' => '0.00'
            ),
            array(
                'name' => 'currency_id',
                'value' => '-99'
            ),
        )
    );

    $createProductBundleResult = call('set_entry',
$createProductBundleParams, $url);

    echo "Create ProductBundles Result<br />";
    echo "<pre>";
    print_r($createProductBundleResult);
    echo "</pre>";

    //relate product to product-bundle
----------------------------------------
    $relationshipProductBundleProductsParams = array(
        'session' => $session_id,
        'module_name' => 'ProductBundles',
        'module_id' => $createProductBundleResult->id,
        'link_field_name' => 'products',
        'related_ids' => array(
            $createProductResult->id
```

```
        ),
    );

    // set the product bundles products relationship
    $relationshipProductBundleProductResult = call('set_relationship',
$relationshipProductBundleProductsParams, $url);

    echo "Create ProductBundleProduct Relationship Result<br />";
    echo "<pre>";
    print_r($relationshipProductBundleProductResult);
    echo "</pre>";

    //relate product-bundle to quote
----------------------------------------
    $relationshipProductBundleQuoteParams = array(
        'session' => $session_id,
        'module_name' => 'Quotes',
        'module_id' => $createQuoteResult->id,
        'link_field_name' => 'product_bundles',
        'related_ids' => array(
            $createProductBundleResult->id
        ),
        'name_value_list' => array()
    );

    // set the product bundles quotes relationship
    $relationshipProductBundleQuoteResult = call('set_relationship',
$relationshipProductBundleQuoteParams, $url);

    echo "Create ProductBundleQuote Relationship Result<br />";
    echo "<pre>";
    print_r($relationshipProductBundleQuoteResult);
    echo "</pre>";
```

Result

```
//Create Quote Result
stdClass Object
(
    [id] => 2e0cd18b-21da-50f0-10f6-517e835a1e09
    [entry_list] => stdClass Object
        (
            [name] => stdClass Object
                (
                    [name] => name
```

```
            [value] => Widget Quote
        )

    [team_count] => stdClass Object
        (
            [name] => team_count
            [value] =>
        )

    [team_name] => stdClass Object
        (
            [name] => team_name
            [value] =>
        )

    [date_quote_expected_closed] => stdClass Object
        (
            [name] => date_quote_expected_closed
            [value] => 2013-05-06
        )

    [quote_stage] => stdClass Object
        (
            [name] => quote_stage
            [value] => Negotiation
        )

    [quote_num] => stdClass Object
        (
            [name] => quote_num
            [value] =>
        )

    [quote_type] => stdClass Object
        (
            [name] => quote_type
            [value] => Quotes
        )

    [subtotal] => stdClass Object
        (
            [name] => subtotal
            [value] => 1230.23
        )

    [subtotal_usdollar] => stdClass Object
```

```
                (
                    [name] => subtotal_usdollar
                    [value] => 1230.23
                )
        )
)

//Create Product Result
stdClass Object
(
    [id] => 6c40f344-a269-d4d0-9929-517e83884fb2
    [entry_list] => stdClass Object
        (
            [name] => stdClass Object
                (
                    [name] => name
                    [value] => Widget
                )

            [quote_id] => stdClass Object
                (
                    [name] => quote_id
                    [value] => 2e0cd18b-21da-50f0-10f6-517e835a1e09
                )

            [status] => stdClass Object
                (
                    [name] => status
                    [value] => Quotes
                )
        )
)

//Create ProductBundles Result
stdClass Object
(
    [id] => a8a4e449-7e72-dea5-9495-517e830a7353
    [entry_list] => stdClass Object
        (
            [name] => stdClass Object
                (
                    [name] => name
                    [value] => Rest SugarOnline Order
                )

            [bundle_stage] => stdClass Object
```

```
                (
                    [name] => bundle_stage
                    [value] => Draft
                )

            [tax] => stdClass Object
                (
                    [name] => tax
                    [value] => 0
                )

            [total] => stdClass Object
                (
                    [name] => total
                    [value] => 0
                )

            [subtotal] => stdClass Object
                (
                    [name] => subtotal
                    [value] => 0
                )

            [currency_id] => stdClass Object
                (
                    [name] => currency_id
                    [value] => -99
                )
        )
)

//Create ProductBundleProduct Relationship Result
stdClass Object
(
    [created] => 1
    [failed] => 0
    [deleted] => 0
)

//Create ProductBundleQuote Relationship Result
stdClass Object
(
    [created] => 1
    [failed] => 0
    [deleted] => 0
)
```

Last Modified: 10/07/2015 02:57pm

Retrieving a List of Fields From a Module

Overview

A PHP example demonstrating how to retrieve fields vardefs from the accounts module with the get_module_fields method using cURL and the v4_1 REST API.

This example will only retrieve the vardefs for the 'id' and 'name' fields.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/rest.php";
    $username = "admin";
    $password = "password";

    function call($method, $parameters, $url)
    {
        ob_start();
        $curl_request = curl_init();

        curl_setopt($curl_request, CURLOPT_URL, $url);
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
        curl_setopt($curl_request, CURLOPT_HEADER, 1);
        curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

        $jsonEncodedData = json_encode($parameters);

        $post = array(
             "method" => $method,
             "input_type" => "JSON",
             "response_type" => "JSON",
             "rest_data" => $jsonEncodedData
        );
```

```
        curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);
        $result = curl_exec($curl_request);
        curl_close($curl_request);

        $result = explode("\r\n\r\n", $result, 2);
        $response = json_decode($result[1]);
        ob_end_flush();

        return $response;
    }

    //login ----------------------------------------
    $login_parameters = array(
         "user_auth" => array(
              "user_name" => $username,
              "password" => md5($password),
              "version" => "1"
         ),
         "application_name" => "RestTest",
         "name_value_list" => array(),
    );

    $login_result = call("login", $login_parameters, $url);

    /*
    echo "<pre>";
    print_r($login_result);
    echo "</pre>";
    */

    //get session id
    $session_id = $login_result->id;

    //retrieve fields --------------------------------
    $get_module_fields_parameters = array(

         //session id
         'session' => $session_id,

         //The name of the module from which to retrieve records
         'module_name' => 'Accounts',

         //Optional. Returns vardefs for the specified fields. An
empty array will return all fields.
         'fields' => array(
```

```
            'id',
            'name',
        ),
    );

    $get_module_fields_result = call("get_module_fields",
$get_module_fields_parameters, $url);

    echo "<pre>";
    print_r($get_module_fields_result);
    echo "</pre>";

?>
```

Result

```
stdClass Object
(
    [module_name] => Accounts
    [table_name] => accounts
    [module_fields] => stdClass Object
        (
            [id] => stdClass Object
                (
                    [name] => id
                    [type] => id
                    [group] =>
                    [id_name] =>
                    [label] => ID
                    [required] => 1
                    [options] => Array
                        (
                        )

                    [related_module] =>
                    [calculated] =>
                    [len] =>
                )

            [name] => stdClass Object
                (
                    [name] => name
                    [type] => name
                    [group] =>
                    [id_name] =>
```

```
                    [label] => Name:
                    [required] => 1
                    [options] => Array
                        (
                        )

                    [related_module] =>
                    [calculated] =>
                    [len] => 150
                )
        )

    [link_fields] => Array
        (
        )
)
```

Last Modified: 09/26/2015 04:23pm

Retrieving a List of Records

Overview

A PHP example demonstrating how to retrieve a list of records from a module with the get_entry_list method using cURL and the v4_1 REST API.

This example will retrieve a list of leads.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/rest.php";
    $username = "admin";
    $password = "password";

    //function to make cURL request
    function call($method, $parameters, $url)
    {
        ob_start();
        $curl_request = curl_init();
```

```
        curl_setopt($curl_request, CURLOPT_URL, $url);
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
        curl_setopt($curl_request, CURLOPT_HEADER, 1);
        curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

        $jsonEncodedData = json_encode($parameters);

        $post = array(
             "method" => $method,
             "input_type" => "JSON",
             "response_type" => "JSON",
             "rest_data" => $jsonEncodedData
        );

        curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);
        $result = curl_exec($curl_request);
        curl_close($curl_request);

        $result = explode("\r\n\r\n", $result, 2);
        $response = json_decode($result[1]);
        ob_end_flush();

        return $response;
    }

    //login ----------------------------------------
    $login_parameters = array(
         "user_auth" => array(
              "user_name" => $username,
              "password" => md5($password),
              "version" => "1"
         ),
         "application_name" => "RestTest",
         "name_value_list" => array(),
    );

    $login_result = call("login", $login_parameters, $url);

    /*
    echo "<pre>";
    print_r($login_result);
    echo "</pre>";
```

```
    */

    //get session id
    $session_id = $login_result->id;

    //get list of records --------------------------------
    $get_entry_list_parameters = array(

         //session id
         'session' => $session_id,

         //The name of the module from which to retrieve records
         'module_name' => 'Leads',

         //The SQL WHERE clause without the word "where".
         'query' => "",

         //The SQL ORDER BY clause without the phrase "order by".
         'order_by' => "",

         //The record offset from which to start.
         'offset' => '0',

         //Optional. A list of fields to include in the results.
         'select_fields' => array(
              'id',
              'name',
              'title',
         ),

         /*
         A list of link names and the fields to be returned for each
link name.
         Example: 'link_name_to_fields_array' => array(array('name' =>
'email_addresses', 'value' => array('id', 'email_address', 'opt_out',
'primary_address')))
         */
         'link_name_to_fields_array' => array(
         ),

         //The maximum number of results to return.
         'max_results' => '2',

         //To exclude deleted records
         'deleted' => '0',
```

```
        //If only records marked as favorites should be returned.
        'Favorites' => false,
    );

    $get_entry_list_result = call('get_entry_list',
$get_entry_list_parameters, $url);

    echo '<pre>';
    print_r($get_entry_list_result);
    echo '</pre>';

?>
```

Result

```
stdClass Object
(
    [result_count] => 2
    [total_count] => 200
    [next_offset] => 2
    [entry_list] => Array
        (
            [0] => stdClass Object
                (
                    [id] => 18124607-69d1-b158-47ff-4f7cb69344f7
                    [module_name] => Leads
                    [name_value_list] => stdClass Object
                        (
                            [id] => stdClass Object
                                (
                                    [name] => id
                                    [value] =>
18124607-69d1-b158-47ff-4f7cb69344f7
                                )

                            [name] => stdClass Object
                                (
                                    [name] => name
                                    [value] => Bernie Worthey
                                )

                            [title] => stdClass Object
                                (
                                    [name] => title
                                    [value] => Senior Product Manager
```

```
                                )
                            )
                    )

                [1] => stdClass Object
                    (
                        [id] => 1cdfddc1-2759-b007-8713-4f7cb64c2e9c
                        [module_name] => Leads
                        [name_value_list] => stdClass Object
                            (
                                [id] => stdClass Object
                                    (
                                        [name] => id
                                        [value] =>
1cdfddc1-2759-b007-8713-4f7cb64c2e9c
                                    )

                                [name] => stdClass Object
                                    (
                                        [name] => name
                                        [value] => Bobbie Kohlmeier
                                    )

                                [title] => stdClass Object
                                    (
                                        [name] => title
                                        [value] => Director Operations
                                    )
                            )
                    )
            )

        [relationship_list] => Array
            (
            )
)
```

Last Modified: 09/26/2015 04:23pm

Retrieving a List of Records With Related Info

Overview

A PHP example demonstrating how to retrieve a list of records with info from a related entity with the get_entry_list method using cURL and the v4_1 REST API.

This example will retrieve a list of contacts and their related email addresses.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/rest.php";
    $username = "admin";
    $password = "password";

    //function to make cURL request
    function call($method, $parameters, $url)
    {
        ob_start();
        $curl_request = curl_init();

        curl_setopt($curl_request, CURLOPT_URL, $url);
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
        curl_setopt($curl_request, CURLOPT_HEADER, 1);
        curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

        $jsonEncodedData = json_encode($parameters);

        $post = array(
             "method" => $method,
             "input_type" => "JSON",
             "response_type" => "JSON",
             "rest_data" => $jsonEncodedData
        );

        curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);
        $result = curl_exec($curl_request);
        curl_close($curl_request);

        $result = explode("\r\n\r\n", $result, 2);
        $response = json_decode($result[1]);
        ob_end_flush();
```

```
    return $response;
}

//login ------------------------------------------------
$login_parameters = array(
     "user_auth" => array(
          "user_name" => $username,
          "password" => md5($password),
          "version" => "1"
     ),
     "application_name" => "RestTest",
     "name_value_list" => array(),
);

$login_result = call("login", $login_parameters, $url);

/*
echo "<pre>";
print_r($login_result);
echo "</pre>";
*/

//get session id
$session_id = $login_result->id;

//retrieve records -------------------------------------
$get_entry_list_parameters = array(

     //session id
     'session' => $session_id,

     //The name of the module from which to retrieve records
     'module_name' => "Contacts",

     //The SQL WHERE clause without the word "where".
     'query' => "",

     //The SQL ORDER BY clause without the phrase "order by".
     'order_by' => "",

     //The record offset from which to start.
     'offset' => "0",

     //Optional. The list of fields to be returned in the results
     'select_fields' => array(
          'id',
```

```
            'first_name',
            'last_name',
        ),

        //A list of link names and the fields to be returned for each
link name
        'link_name_to_fields_array' => array(
            array(
                'name' => 'email_addresses',
                'value' => array(
                    'id',
                    'email_address',
                    'opt_out',
                    'primary_address'
                ),
            ),
        ),

        //The maximum number of results to return.
        'max_results' => '2',

        //To exclude deleted records
        'deleted' => 0,

        //If only records marked as favorites should be returned.
        'Favorites' => false,

    );

    $get_entry_list_result = call("get_entry_list",
$get_entry_list_parameters, $url);

    echo "<pre>";
    print_r($get_entry_list_result);
    echo "</pre>";

?>
```

Result

```
stdClass Object
(
   [result_count] => 2
   [total_count] => 200
   [next_offset] => 2
```

```
[entry_list] => Array
(
   [0] => stdClass Object
   (
      [id] => 116d9bc6-4a24-b826-952e-4f7cb6b25ea7
      [module_name] => Contacts
      [name_value_list] => stdClass Object
      (
         [id] => stdClass Object
         (
            [name] => id
            [value] => 116d9bc6-4a24-b826-952e-4f7cb6b25ea7
         )

         [first_name] => stdClass Object
         (
            [name] => first_name
            [value] => Lucinda
         )

         [last_name] => stdClass Object
         (
            [name] => last_name
            [value] => Jacoby
         )
      )
   )

   [1] => stdClass Object
   (
      [id] => 11c263ef-ff61-71ff-f090-4f7cb6fc9f68
      [module_name] => Contacts
      [name_value_list] => stdClass Object
      (
         [id] => stdClass Object
         (
            [name] => id
            [value] => 11c263ef-ff61-71ff-f090-4f7cb6fc9f68
         )

         [first_name] => stdClass Object
         (
            [name] => first_name
            [value] => Ike
         )
```

```
            [last_name] => stdClass Object
            (
               [name] => last_name
               [value] => Gassaway
            )
         )
      )
   )

   [relationship_list] => Array
   (
      [0] => stdClass Object
      (
         [link_list] => Array
         (
            [0] => stdClass Object
            (
               [name] => email_addresses
               [records] => Array
               (
                  [0] => stdClass Object
                  (
                     [link_value] => stdClass Object
                     (
                        [id] => stdClass Object
                        (
                           [name] => id
                           [value] =>
13066f13-d6ea-405c-0f95-4f7cb6fa3a08
                        )

                        [email_address] => stdClass Object
                        (
                           [name] => email_address
                           [value] => support52@example.org
                        )

                        [opt_out] => stdClass Object
                        (
                           [name] => opt_out
                           [value] => 0
                        )

                        [primary_address] => stdClass Object
                        (
                           [name] => primary_address
```

```
                           [value] =>
                        )
                     )
                  )

                  [1] => stdClass Object
                  (
                     [link_value] => stdClass Object
                     (
                        [id] => stdClass Object
                        (
                           [name] => id
                           [value] =>
13e3d111-b226-c363-4832-4f7cb699a3a0
                        )

                        [email_address] => stdClass Object
                        (
                           [name] => email_address
                           [value] => qa.section@example.it
                        )

                        [opt_out] => stdClass Object
                        (
                           [name] => opt_out
                           [value] => 1
                        )

                        [primary_address] => stdClass Object
                        (
                           [name] => primary_address
                           [value] =>
                        )
                     )
                  )
               )
            )
         )
      )

      [1] => stdClass Object
      (
         [link_list] => Array
         (
            [0] => stdClass Object
            (
```

```
                [name] => email_addresses
                [records] => Array
                (
                   [0] => stdClass Object
                   (
                      [link_value] => stdClass Object
                      (
                         [id] => stdClass Object
                         (
                            [name] => id
                            [vale] =>
16aeaf8b-b31f-943d-3844-4f7cb6ac63f2
                         )

                         [email_address] => stdClass Object
                         (
                            [name] => email_address
                            [value] => vegan.sales.im@example.cn
                         )

                         [opt_out] => stdClass Object
                         (
                            [name] => opt_out
                            [value] => 0
                         )

                         [primary_address] => stdClass Object
                         (
                            [name] => primary_address
                            [value] =>
                         )
                      )
                   )

                   [1] => stdClass Object
                   (
                      [link_value] => stdClass Object
                      (
                         [id] => stdClass Object
                         (
                            [name] => id
                            [value] =>
173bacf5-5ea6-3906-d91d-4f7cb6c6e883
                         )

                         [email_address] => stdClass Object
```

```
                        (
                            [name] => email_address
                            [value] => kid.the.section@example.org
                        )

                        [opt_out] => stdClass Object
                        (
                            [name] => opt_out
                            [value] => 1
                        )

                        [primary_address] => stdClass Object
                        (
                            [name] => primary_address
                            [value] =>
                        )
                    )
                )
            )
        )
    )
)
```

Last Modified: 09/26/2015 04:23pm

Retrieving Email Attachments

Overview

A PHP example demonstrating how to retrieve an email and its attachments from using the get_entry and get_note_attachment methods using cURL and the v4_1 REST API.

This example will retrieve a specified email record by its ID and write the attachments to the local filesystem.

Example

```
<?php
```

```
$url = "http://{site_url}/service/v4_1/rest.php";
$username = "admin";
$password = "password";

//function to make cURL request
function call($method, $parameters, $url)
{
    ob_start();

    $curl_request = curl_init();
    curl_setopt($curl_request, CURLOPT_URL, $url);
    curl_setopt($curl_request, CURLOPT_POST, 1);
    curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
    curl_setopt($curl_request, CURLOPT_HEADER, 1);
    curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
    curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

    $jsonEncodedData = json_encode($parameters);

    $post = array(
        "method" => $method,
        "input_type" => "JSON",
        "response_type" => "JSON",
        "rest_data" => $jsonEncodedData
    );

    curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);

    $result = curl_exec($curl_request);

    curl_close($curl_request);

    $result = explode("\r\n\r\n", $result, 2);
    $response = json_decode($result[1]);

    ob_end_flush();
    return $response;
}

//login ------------------------------------------
$login_parameters = array(
    "user_auth" => array(
        "user_name" => $username,
        "password" => md5($password),
```

```
        "version" => "1"
    ),

    "application_name" => "RestTest",
    "name_value_list" => array(),
);

$login_result = call("login", $login_parameters, $url);

/*
echo "<pre>";
print_r($login_result);
echo "</pre>";
*/

//get session id
$session_id = $login_result->id;

//retrieve the email ------------------------------------------------
// email id of an email with an attachment
$email_id   = '5826bd75-527a-a736-edf5-5205421467bf';

// use get_entry to get the email contents
$get_entry_parameters = array(
    'session' => $session_id,
    'module_name' => 'Emails',
    'id' => $email_id,
    'select_fields' => array(),
    'link_name_to_fields_array' => array(
        array(
            'name' => 'notes',
            'value' => array(
                'id',
                'name',
                'file_mime_type',
                'filename',
                'description',
            ),
        ),
    ),
    'track_view' => false
);

$get_entry_result = call('get_entry', $get_entry_parameters, $url);

//Email record contents
```

```
echo "<pre>";
print_r($get_entry_result);
echo "</pre>";

if (!isset($get_entry_result->entry_list[0]))
{
    echo "Email not found!";
    die();
}

if (!isset($get_entry_result->relationship_list) ||
count($get_entry_result->relationship_list) == 0)
{
    echo "No attachments found!";
    die();
}

//retrieve any attachments
-------------------------------------------
foreach ($get_entry_result->relationship_list[0][0]->records as $key
=> $attachmentInfo)
{
    $get_note_attachment_parameters = array(
        'session' => $session_id,
        'id'      => $attachmentInfo->id->value,
    );

    $get_note_attachment_result = call('get_note_attachment',
$get_note_attachment_parameters, $url);

    //attachment contents
    echo "<pre>";

    print_r($get_note_attachment_result);
    echo "</pre>";

    $file_name =
$get_note_attachment_result->note_attachment->filename;

    //decode and get file contents
    $file_contents =
base64_decode($get_note_attachment_result->note_attachment->file);

    //write file
    file_put_contents($file_name, $file_contents);
}
```

Result

```
//Email Result
stdClass Object
(
    [entry_list] => Array
        (
            [0] => stdClass Object
                (
                    [id] => 5826bd75-527a-a736-edf5-5205421467bf
                    [module_name] => Emails
                    [name_value_list] => stdClass Object
                        (
                            [assigned_user_name] => stdClass Object
                                (
                                    [name] => assigned_user_name
                                    [value] => Administrator
                                )

                            [modified_by_name] => stdClass Object
                                (
                                    [name] => modified_by_name
                                    [value] => Administrator
                                )

                            [created_by_name] => stdClass Object
                                (
                                    [name] => created_by_name
                                    [value] => Administrator
                                )

                            [team_id] => stdClass Object
                                (
                                    [name] => team_id
                                    [value] => 1
                                )

                            [team_set_id] => stdClass Object
                                (
                                    [name] => team_set_id
                                    [value] => 1
                                )

                            [team_name] => stdClass Object
```

```
                                    (
                                        [name] => team_name
                                        [value] => Global
                                    )

                                [id] => stdClass Object
                                    (
                                        [name] => id
                                        [value] =>
5826bd75-527a-a736-edf5-5205421467bf
                                    )

                                [date_entered] => stdClass Object
                                    (
                                        [name] => date_entered
                                        [value] => 2013-08-09 19:28:00
                                    )

                                [date_modified] => stdClass Object
                                    (
                                        [name] => date_modified
                                        [value] => 2013-08-09 19:29:08
                                    )

                                [assigned_user_id] => stdClass Object
                                    (
                                        [name] => assigned_user_id
                                        [value] => 1
                                    )

                                [modified_user_id] => stdClass Object
                                    (
                                        [name] => modified_user_id
                                        [value] => 1
                                    )

                                [created_by] => stdClass Object
                                    (
                                        [name] => created_by
                                        [value] => 1
                                    )

                                [deleted] => stdClass Object
                                    (
                                        [name] => deleted
                                        [value] => 0
```

```
    )

[from_addr_name] => stdClass Object
    (
        [name] => from_addr_name
        [value] => SugarCRM
    )

[to_addrs_names] => stdClass Object
    (
        [name] => to_addrs_names
        [value] => email@address.com
    )

[description_html] => stdClass Object
    (
        [name] => description_html
        [value] =>
    )

[description] => stdClass Object
    (
        [name] => description
        [value] =>
    )

[date_sent] => stdClass Object
    (
        [name] => date_sent
        [value] => 2013-08-09 19:28:00
    )

[message_id] => stdClass Object
    (
        [name] => message_id
        [value] =>
    )

[message_uid] => stdClass Object
    (
        [name] => message_uid
        [value] =>
    )

[name] => stdClass Object
    (
```

```
                [name] => name
                [value] => Example
            )

        [type] => stdClass Object
            (
                [name] => type
                [value] => out
            )

        [status] => stdClass Object
            (
                [name] => status
                [value] => read
            )

        [flagged] => stdClass Object
            (
                [name] => flagged
                [value] => 0
            )

        [reply_to_status] => stdClass Object
            (
                [name] => reply_to_status
                [value] => 0
            )

        [intent] => stdClass Object
            (
                [name] => intent
                [value] => pick
            )

        [mailbox_id] => stdClass Object
            (
                [name] => mailbox_id
                [value] =>
            )

        [parent_name] => stdClass Object
            (
                [name] => parent_name
                [value] => Having trouble adding
new items
            )
```

```
                                [parent_type] => stdClass Object
                                    (
                                        [name] => parent_type
                                        [value] => Cases
                                    )

                                [parent_id] => stdClass Object
                                    (
                                        [name] => parent_id
                                        [value] =>
116d4d46-928c-d4af-c22e-518ae4eb13fc
                                    )
                            )
                    )
            )

        [relationship_list] => Array
            (
                [0] => Array
                    (
                        [0] => stdClass Object
                            (
                                [name] => notes
                                [records] => Array
                                    (
                                        [0] => stdClass Object
                                            (
                                                [id] => stdClass Object
                                                    (
                                                        [name] => id
                                                        [value] =>
1b63a8f9-ce67-6aad-b5a4-52054af18c47
                                                    )

                                                [name] => stdClass Object
                                                    (
                                                        [name] => name
                                                        [value] =>
Example.zip
                                                    )

                                                [file_mime_type] =>
stdClass Object
                                                    (
                                                        [name] =>
```

```
file_mime_type
                                                    [value] =>
application/zip
                                                )

                                            [filename] => stdClass
Object
                                                (
                                                    [name] => filename
                                                    [value] =>
Example2.zip
                                                )

                                            [description] => stdClass
Object
                                                (
                                                    [name] =>
description
                                                    [value] =>
                                                )

                                            [_empty_] => stdClass
Object
                                                (
                                                    [name] =>
                                                    [value] =>
                                                )
                                        )

                                    [1] => stdClass Object
                                        (
                                            [id] => stdClass Object
                                                (
                                                    [name] => id
                                                    [value] =>
592f382d-b633-fd5f-803e-5205423a6d0b
                                                )

                                            [name] => stdClass Object
                                                (
                                                    [name] => name
                                                    [value] =>
Example2.zip
                                                )

                                            [file_mime_type] =>
```

```
stdClass Object
                                                    (
                                                        [name] =>
file_mime_type
                                                        [value] =>
application/zip
                                                    )

                                                [filename] => stdClass
Object
                                                    (
                                                        [name] => filename
                                                        [value] =>
Example2.zip
                                                    )

                                                [description] => stdClass
Object
                                                    (
                                                        [name] =>
description
                                                        [value] =>
                                                    )

                                                [_empty_] => stdClass
Object
                                                    (
                                                        [name] =>
                                                        [value] =>
                                                    )
                                            )
                                    )
                            )
                )
        )
)

//Attachment 1 Result
stdClass Object
(
    [note_attachment] => stdClass Object
        (
            [id] => 1b63a8f9-ce67-6aad-b5a4-52054af18c47
            [filename] => Example.zip
            [file] =>
UEsDBAoAAAAIAOujCEOkMbvsNa0AAMCFAgAXAAAARmlsZXMvaW
```

```
            [related_module_id] =>
5826bd75-527a-a736-edf5-5205421467bf
            [related_module_name] => Emails
        )
)

//Attachment 2 Result
stdClass Object
(
    [note_attachment] => stdClass Object
        (
            [id] => 592f382d-b633-fd5f-803e-5205423a6d0b
            [filename] => Example2.zip
            [file] =>
AEUoaAAARmlslsZXMvaWZXujCEOkMbvsNa0AAMCFAgAXAAAARm
            [related_module_id] =>
5826bd75-527a-a736-edf5-5205421467bf
            [related_module_name] => Emails
        )
)
```

Last Modified: 09/26/2015 04:23pm

Retrieving Multiple Records by ID

Overview

A PHP example demonstrating how to retrieve multiple records from the accounts module with the get_entries method using cURL and the v4_1 REST API.

This example will only retrieve 2 records and return the following fields: 'name', 'billing_address_state' and 'billing_address_country'.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/rest.php";
    $username = "admin";
    $password = "password";

    //function to make cURL request
```

```
    function call($method, $parameters, $url)
    {
        ob_start();
        $curl_request = curl_init();

        curl_setopt($curl_request, CURLOPT_URL, $url);
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
        curl_setopt($curl_request, CURLOPT_HEADER, 1);
        curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

        $jsonEncodedData = json_encode($parameters);

        $post = array(
             "method" => $method,
             "input_type" => "JSON",
             "response_type" => "JSON",
             "rest_data" => $jsonEncodedData
        );

        curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);
        $result = curl_exec($curl_request);
        curl_close($curl_request);

        $result = explode("\r\n\r\n", $result, 2);
        $response = json_decode($result[1]);
        ob_end_flush();

        return $response;
    }

    //login ------------------------------------------
    $login_parameters = array(
         "user_auth" => array(
              "user_name" => $username,
              "password" => md5($password),
              "version" => "1"
         ),
         "application_name" => "RestTest",
         "name_value_list" => array(),
    );

    $login_result = call("login", $login_parameters, $url);
```

```
    /*
    echo "<pre>";
    print_r($login_result);
    echo "</pre>";
    */

    //get session id
    $session_id = $login_result->id;

    //retrieve records ----------------------------------
    $get_entries_parameters = array(

         //session id
         'session' => $session_id,

         //The name of the module from which to retrieve records
         'module_name' => 'Accounts',

         //An array of SugarBean IDs
         'ids' => array(
             '20328809-9d0a-56fc-0e7c-4f7cb6eb1c83',
             '328b22a6-d784-66d9-0295-4f7cb59e8cbb',
         ),

         //Optional. The list of fields to be returned in the results
         'select_fields' => array(
            'name',
            'billing_address_state',
            'billing_address_country'
         ),

         //A list of link names and the fields to be returned for each
link name
         'link_name_to_fields_array' => array(
         ),
    );

    $get_entries_result = call("get_entries", $get_entries_parameters,
$url);

    echo "<pre>";
    print_r($get_entries_result);
    echo "</pre>";

?>
```

Result

```
stdClass Object
(
    [entry_list] => Array
        (
            [0] => stdClass Object
                (
                    [id] => 20328809-9d0a-56fc-0e7c-4f7cb6eb1c83
                    [module_name] => Accounts
                    [name_value_list] => stdClass Object
                        (
                            [name] => stdClass Object
                                (
                                    [name] => name
                                    [value] => Jungle Systems Inc
                                )

                            [billing_address_state] => stdClass Object
                                (
                                    [name] => billing_address_state
                                    [value] => NY
                                )

                            [billing_address_country] => stdClass
Object
                                (
                                    [name] => billing_address_country
                                    [value] => USA
                                )
                        )
                )

            [1] => stdClass Object
                (
                    [id] => 328b22a6-d784-66d9-0295-4f7cb59e8cbb
                    [module_name] => Accounts
                    [name_value_list] => stdClass Object
                        (
                            [name] => stdClass Object
                                (
                                    [name] => name
                                    [value] => Riviera Hotels
                                )
```

```
                            [billing_address_state] => stdClass Object
                                (
                                    [name] => billing_address_state
                                    [value] => CA
                                )

                            [billing_address_country] => stdClass
Object
                                (
                                    [name] => billing_address_country
                                    [value] => USA
                                )
                        )
                )
        )

    [relationship_list] => Array
        (
        )
)
```

Last Modified: 09/26/2015 04:23pm

Retrieving Records by Email Domain

Overview

A PHP example demonstrating how to retrieve email addresses based on an email domain with the search_by_module and get_entries methods using cURL and the v4_1 REST API.

When using the search_by_module method, the email address information is not returned in the result. Due to this behavior, we will gather the record ids and pass them back to the get_entries method to fetch our related email addresses.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/rest.php";
    $username = "admin";
```

```
    $password = "password";

    //function to make cURL request
    function call($method, $parameters, $url)
    {
        ob_start();
        $curl_request = curl_init();

        curl_setopt($curl_request, CURLOPT_URL, $url);
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
        curl_setopt($curl_request, CURLOPT_HEADER, 1);
        curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

        $jsonEncodedData = json_encode($parameters);

        $post = array(
            "method" => $method,
            "input_type" => "JSON",
            "response_type" => "JSON",
            "rest_data" => $jsonEncodedData
        );

        curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);
        $result = curl_exec($curl_request);
        curl_close($curl_request);

        $result = explode("\r\n\r\n", $result, 2);
        $response = json_decode($result[1]);
        ob_end_flush();

        return $response;
    }

    //login ------------------------------------------
    $login_parameters = array(
        "user_auth" => array(
            "user_name" => $username,
            "password" => md5($password),
            "version" => "1"
        ),
        "application_name" => "RestTest",
        "name_value_list" => array(),
```

```
    );

    $login_result = call("login", $login_parameters, $url);

    /*
    echo "<pre>";
    print_r($login_result);
    echo "</pre>";
    */

    //get session id
    $session_id = $login_result->id;

    //search_by_module
-------------------------------------------------
    $search_by_module_parameters = array(
        "session" => $session_id,
        'search_string' => '%@example.com',
        'modules' => array(
            'Accounts',
            'Contacts',
            'Leads',
        ),
        'offset' => 0,
        'max_results' => 1,
        'assigned_user_id' => '',
        'select_fields' => array('id'),
        'unified_search_only' => false,
        'favorites' => false
    );

    $search_by_module_results = call('search_by_module',
$search_by_module_parameters, $url);

    /*
    echo '<pre>';
    print_r($search_by_module_results);
    echo '</pre>';
    */

    $record_ids = array();
    foreach ($search_by_module_results->entry_list as $results)
    {
        $module = $results->name;

        foreach ($results->records as $records)
```

```
        {
            foreach($records as $record)
            {
                if ($record->name = 'id')
                {
                    $record_ids[$module][] = $record->value;
                    //skip any additional fields
                    break;
                }
            }
        }

    }

    $get_entries_results = array();
    $modules = array_keys($record_ids);

    foreach($modules as $module)
    {
        $get_entries_parameters = array(
            //session id
            'session' => $session_id,

            //The name of the module from which to retrieve records
            'module_name' => $module,

            //An array of record IDs
            'ids' => $record_ids[$module],

            //The list of fields to be returned in the results
            'select_fields' => array(
                'name',
            ),

            //A list of link names and the fields to be returned for
each link name
            'link_name_to_fields_array' => array(
                array(
                    'name' => 'email_addresses',
                    'value' => array(
                        'email_address',
                        'opt_out',
                        'primary_address'
                    ),
                ),
            ),
```

```
            //Flag the record as a recently viewed item
            'track_view' => false,
        );

        $get_entries_results[$module] = call('get_entries',
$get_entries_parameters, $url);
    }

    echo '<pre>';
    print_r($get_entries_results);
    echo '</pre>';
```

Result

```
Array
(
   [Accounts] => stdClass Object
   (
      [entry_list] => Array
      (
         [0] => stdClass Object
         (
            [id] => 1bb7ef28-64b9-cbd5-e7d6-5282a3b96953
            [module_name] => Accounts
            [name_value_list] => stdClass Object
            (
               [name] => stdClass Object
               (
                  [name] => name
                  [value] => Underwater Mining Inc.
               )
            )
         )

         [1] => stdClass Object
         (
            [id] => efbc0c4e-cc72-f843-b6ed-5282a38dad7f
            [module_name] => Accounts
            [name_value_list] => stdClass Object
            (
               [name] => stdClass Object
               (
                  [name] => name
```

```
            [value] => 360 Vacations
         )
      )
   )
)

[relationship_list] => Array
(
   [0] => stdClass Object
   (
      [link_list] => Array
      (
         [0] => stdClass Object
         (
            [name] => email_addresses
            [records] => Array
            (
               [0] => stdClass Object
               (
                  [link_value] => stdClass Object
                  (
                     [email_address] => stdClass Object
                     (
                        [name] => email_address
                        [value] => beans.the.qa@example.com
                     )

                     [opt_out] => stdClass Object
                     (
                        [name] => opt_out
                        [value] => 0
                     )

                     [primary_address] => stdClass Object
                     (
                        [name] => primary_address
                        [value] =>
                     )
                  )
               )

               [1] => stdClass Object
               (
                  [link_value] => stdClass Object
                  (
                     [email_address] => stdClass Object
```

```
                    (
                        [name] => email_address
                        [value] => section.sales@example.edu
                    )

                    [opt_out] => stdClass Object
                    (
                        [name] => opt_out
                        [value] => 0
                    )

                    [primary_address] => stdClass Object
                    (
                        [name] => primary_address
                        [value] =>
                    )
                )
            )
        )
    )
)

[1] => stdClass Object
(
    [link_list] => Array
    (
        [0] => stdClass Object
        (
            [name] => email_addresses
            [records] => Array
            (
                [0] => stdClass Object
                (
                    [link_value] => stdClass Object
                    (
                        [email_address] => stdClass Object
                        (
                            [name] => email_address
                            [value] => section51@example.com
                        )

                        [opt_out] => stdClass Object
                        (
                            [name] => opt_out
                            [value] => 0
```

```
                        )

                        [primary_address] => stdClass Object
                        (
                            [name] => primary_address
                            [value] =>
                        )
                    )
                )

                [1] => stdClass Object
                (
                    [link_value] => stdClass Object
                    (
                        [email_address] => stdClass Object
                        (
                            [name] => email_address
                            [value] => kid.sugar.qa@example.co.uk
                        )

                        [opt_out] => stdClass Object
                        (
                            [name] => opt_out
                            [value] => 0
                        )

                        [primary_address] => stdClass Object
                        (
                            [name] => primary_address
                            [value] =>
                        )
                    )
                )
            )
        )
    )
)

[Contacts] => stdClass Object
(
    [entry_list] => Array
    (
        [0] => stdClass Object
        (
```

```
            [id] => 24330979-0e39-f9ec-2622-5282a372f39b
            [module_name] => Contacts
            [name_value_list] => stdClass Object
            (
                [name] => stdClass Object
                (
                    [name] => name
                    [value] => Errol Goldberg
                )
            )
        )

        [1] => stdClass Object
        (
            [id] => 2542dad2-85f3-5529-3a30-5282a3104e31
            [module_name] => Contacts
            [name_value_list] => stdClass Object
            (
                [name] => stdClass Object
                (
                    [name] => name
                    [value] => Luis Deegan
                )
            )
        )
    )

    [relationship_list] => Array
    (
        [0] => stdClass Object
        (
            [link_list] => Array
            (
                [0] => stdClass Object
                (
                    [name] => email_addresses
                    [records] => Array
                    (
                        [0] => stdClass Object
                        (
                            [link_value] => stdClass Object
                            (
                                [email_address] => stdClass Object
                                (
                                    [name] => email_address
                                    [value] => hr.phone@example.com
```

```
                    )

                    [opt_out] => stdClass Object
                    (
                        [name] => opt_out
                        [value] => 0
                    )

                    [primary_address] => stdClass Object
                    (
                       [name] => primary_address
                       [value] =>
                    )
                )
            )

            [1] => stdClass Object
            (
               [link_value] => stdClass Object
               (
                    [email_address] => stdClass Object
                    (
                       [name] => email_address
                       [value] => the.info.phone@example.cn
                    )

                    [opt_out] => stdClass Object
                    (
                       [name] => opt_out
                       [value] => 1
                    )

                    [primary_address] => stdClass Object
                    (
                       [name] => primary_address
                       [value] =>
                    )
               )
            )
        )
    )
)

[1] => stdClass Object
(
```

```
[link_list] => Array
(
   [0] => stdClass Object
   (
      [name] => email_addresses
      [records] => Array
      (
         [0] => stdClass Object
         (
            [link_value] => stdClass Object
            (
               [email_address] => stdClass Object
               (
                  [name] => email_address
                  [value] => im.the.sales@example.name
               )

               [opt_out] => stdClass Object
               (
                  [name] => opt_out
                  [value] => 0
               )

               [primary_address] => stdClass Object
               (
                  [name] => primary_address
                  [value] =>
               )
            )
         )

         [1] => stdClass Object
         (
            [link_value] => stdClass Object
            (
               [email_address] => stdClass Object
               (
                  [name] => email_address
                  [value] => the36@example.com
               )

               [opt_out] => stdClass Object
               (
                  [name] => opt_out
                  [value] => 1
               )
```

```
                        [primary_address] => stdClass Object
                        (
                           [name] => primary_address
                           [value] =>
                        )
                     )
                  )
               )
            )
         )
      )
   )

   [Leads] => stdClass Object
   (
      [entry_list] => Array
      (
         [0] => stdClass Object
         (
            [id] => 83abeeff-5e71-0f06-2e5f-5282a3f04f41
            [module_name] => Leads
            [name_value_list] => stdClass Object
            (
               [name] => stdClass Object
               (
                  [name] => name
                  [value] => Caryn Wert
               )
            )
         )

         [1] => stdClass Object
         (
            [id] => 2a3b304b-5167-8432-d4e7-5282a300eabb
            [module_name] => Leads
            [name_value_list] => stdClass Object
            (
               [name] => stdClass Object
               (
                  [name] => name
                  [value] => Marco Castonguay
               )
            )
         )
```

```
)

[relationship_list] => Array
(
   [0] => stdClass Object
   (
      [link_list] => Array
      (
         [0] => stdClass Object
         (
            [name] => email_addresses
            [records] => Array
            (
               [0] => stdClass Object
               (
                  [link_value] => stdClass Object
                  (
                     [email_address] => stdClass Object
                     (
                        [name] => email_address
                        [value] => hr60@example.com
                     )

                     [opt_out] => stdClass Object
                     (
                        [name] => opt_out
                        [value] => 0
                     )

                     [primary_address] => stdClass Object
                     (
                        [name] => primary_address
                        [value] =>
                     )
                  )
               )
            )
         )
      )
   )

   [1] => stdClass Object
   (
      [link_list] => Array
      (
         [0] => stdClass Object
```

```
                (
                    [name] => email_addresses
                    [records] => Array
                    (
                        [0] => stdClass Object
                        (
                            [link_value] => stdClass Object
                            (
                                [email_address] => stdClass Object
                                (
                                    [name] => email_address
                                    [value] => im.the@example.com
                                )

                                [opt_out] => stdClass Object
                                (
                                    [name] => opt_out
                                    [value] => 0
                                )

                                [primary_address] => stdClass Object
                                (
                                    [name] => primary_address
                                    [value] =>
                                )
                            )
                        )
                    )
                )
            )
        )
    )
)
```

Last Modified: 09/26/2015 04:23pm

Retrieving Related Records

Overview

A PHP example demonstrating how to retrieve a list of related records with the get_relationships method using cURL and the v4_1 REST API.

This example will retrieve a list of leads related to a specific target list.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/rest.php";
    $username = "admin";
    $password = "password";

    //function to make cURL request
    function call($method, $parameters, $url)
    {
        ob_start();
        $curl_request = curl_init();

        curl_setopt($curl_request, CURLOPT_URL, $url);
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
        curl_setopt($curl_request, CURLOPT_HEADER, 1);
        curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

        $jsonEncodedData = json_encode($parameters);

        $post = array(
             "method" => $method,
             "input_type" => "JSON",
             "response_type" => "JSON",
             "rest_data" => $jsonEncodedData
        );

        curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);
        $result = curl_exec($curl_request);
        curl_close($curl_request);

        $result = explode("\r\n\r\n", $result, 2);
        $response = json_decode($result[1]);
        ob_end_flush();

        return $response;
    }
```

```
    //login ------------------------------------------
    $login_parameters = array(
         "user_auth" => array(
              "user_name" => $username,
              "password" => md5($password),
              "version" => "1"
         ),
         "application_name" => "RestTest",
         "name_value_list" => array(),
    );

    $login_result = call("login", $login_parameters, $url);

    /*
    echo "<pre>";
    print_r($login_result);
    echo "</pre>";
    */

    //get session id
    $session_id = $login_result->id;

    //retrieve related list ------------------------------
    $get_relationships_parameters = array(

         'session'=>$session_id,

         //The name of the module from which to retrieve records.
         'module_name' => 'ProspectLists',

         //The ID of the specified module bean.
         'module_id' => '76d0e694-ef66-ddd5-9bdf-4febd3af44d5',

         //The relationship name of the linked field from which to
return records.
         'link_field_name' => 'leads',

         //The portion of the WHERE clause from the SQL statement used
to find the related items.
         'related_module_query' => '',

         //The related fields to be returned.
         'related_fields' => array(
            'id',
            'first_name',
            'last_name',
```

```
        ),

        //For every related bean returned, specify link field names
to field information.
        'related_module_link_name_to_fields_array' => array(
        ),

        //To exclude deleted records
        'deleted'=> '0',

        //order by
        'order_by' => '',

        //offset
        'offset' => 0,

        //limit
        'limit' => 5,
    );

    $get_relationships_result = call("get_relationships",
$get_relationships_parameters, $url);

    echo "<pre>";
    print_r($get_relationships_result);
    echo "</pre>";

?>
```

Results

```
stdClass Object
(
    [entry_list] => Array
        (
            [0] => stdClass Object
                (
                    [id] => 117c26c0-11d4-7b8b-cb8f-4f7cb6823dd8
                    [module_name] => Leads
                    [name_value_list] => stdClass Object
                        (
                            [id] => stdClass Object
                                (
                                    [name] => id
                                    [value] =>
```

```
117c26c0-11d4-7b8b-cb8f-4f7cb6823dd8
                                )

                            [first_name] => stdClass Object
                                (
                                    [name] => first_name
                                    [value] => Diane
                                )

                            [last_name] => stdClass Object
                                (
                                    [name] => last_name
                                    [value] => Mckamey
                                )
                        )
                )

            [1] => stdClass Object
                (
                    [id] => 142faeef-1a19-b53a-b780-4f7cb600c553
                    [module_name] => Leads
                    [name_value_list] => stdClass Object
                        (
                            [id] => stdClass Object
                                (
                                    [name] => id
                                    [value] =>
142faeef-1a19-b53a-b780-4f7cb600c553
                                )

                            [first_name] => stdClass Object
                                (
                                    [name] => first_name
                                    [value] => Leonor
                                )

                            [last_name] => stdClass Object
                                (
                                    [name] => last_name
                                    [value] => Reser
                                )
                        )
                )
        )

    [relationship_list] => Array
```

```
        (
        )
)
```

Last Modified: 09/26/2015 04:23pm

Searching Records

Overview

A PHP example demonstrating how to search the accounts module with the search_by_module method using cURL and the v4_1 REST API.

This script will return two results, sorted by the id field, and return the value of the id, name, account_type, phone_office, and assigned_user_name fields.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/rest.php";
    $username = "admin";
    $password = "password";

    //function to make cURL request
    function call($method, $parameters, $url)
    {
        ob_start();
        $curl_request = curl_init();

        curl_setopt($curl_request, CURLOPT_URL, $url);
        curl_setopt($curl_request, CURLOPT_POST, 1);
        curl_setopt($curl_request, CURLOPT_HTTP_VERSION,
CURL_HTTP_VERSION_1_0);
        curl_setopt($curl_request, CURLOPT_HEADER, 1);
        curl_setopt($curl_request, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($curl_request, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($curl_request, CURLOPT_FOLLOWLOCATION, 0);

        $jsonEncodedData = json_encode($parameters);

        $post = array(
```

```
        "method" => $method,
        "input_type" => "JSON",
        "response_type" => "JSON",
        "rest_data" => $jsonEncodedData
    );

    curl_setopt($curl_request, CURLOPT_POSTFIELDS, $post);
    $result = curl_exec($curl_request);
    curl_close($curl_request);

    $result = explode("\r\n\r\n", $result, 2);
    $response = json_decode($result[1]);
    ob_end_flush();

    return $response;
}

//login ----------------------------------------
$login_parameters = array(
     "user_auth" => array(
          "user_name" => $username,
          "password" => md5($password),
          "version" => "1"
     ),
     "application_name" => "RestTest",
     "name_value_list" => array(),
);

$login_result = call("login", $login_parameters, $url);

/*
echo "<pre>";
print_r($login_result);
echo "</pre>";
*/

//get session id
$session_id = $login_result->id;

//search ----------------------------------------
$search_by_module_parameters = array(
    //Session id
    "session" => $session_id,

    //The string to search for.
    'search_string' => 'Customer',
```

```
        //The list of modules to query.
        'modules' => array(
        'Accounts',
        ),

        //The record offset from which to start.
        'offset' => 0,

        //The maximum number of records to return.
        'max_results' => 2,

        //Filters records by the assigned user ID.
        //Leave this empty if no filter should be applied.
        'id' => '',

        //An array of fields to return.
        //If empty the default return fields will be from the active
listviewdefs.
        'select_fields' => array(
            'id',
            'name',
            'account_type',
            'phone_office',
            'assigned_user_name',
        ),

        //If the search is to only search modules participating in the
unified search.
        //Unified search is the SugarCRM Global Search alternative to
Full-Text Search.
        'unified_search_only' => false,

        //If only records marked as favorites should be returned.
        'favorites' => false
    );

    $search_by_module_result = call('search_by_module',
$search_by_module_parameters, $url);

    echo '<pre>';
    print_r($search_by_module_result);
    echo '</pre>';

?>
```

Result

```
stdClass Object
(
    [result_count] => 2
    [total_count] => 200
    [next_offset] => 2
    [entry_list] => Array
        (
            [0] => stdClass Object
                (
                    [id] => 18124607-69d1-b158-47ff-4f7cb69344f7
                    [module_name] => Leads
                    [name_value_list] => stdClass Object
                        (
                            [id] => stdClass Object
                                (
                                    [name] => id
                                    [value] =>
18124607-69d1-b158-47ff-4f7cb69344f7
                                )

                            [name] => stdClass Object
                                (
                                    [name] => name
                                    [value] => Bernie Worthey
                                )

                            [title] => stdClass Object
                                (
                                    [name] => title
                                    [value] => Senior Product Manager
                                )
                        )
                )

            [1] => stdClass Object
                (
                    [id] => 1cdfddc1-2759-b007-8713-4f7cb64c2e9c
                    [module_name] => Leads
                    [name_value_list] => stdClass Object
                        (
                            [id] => stdClass Object
                                (
                                    [name] => id
                                    [value] =>
```

```
1cdfddc1-2759-b007-8713-4f7cb64c2e9c
                                    )

                                [name] => stdClass Object
                                    (
                                        [name] => name
                                        [value] => Bobbie Kohlmeier
                                    )

                                [title] => stdClass Object
                                    (
                                        [name] => title
                                        [value] => Director Operations
                                    )
                            )
                    )
            )

        [relationship_list] => Array
            (
            )
)
```

Last Modified: 09/26/2015 04:23pm

SOAP

Examples of v4_1 SOAP API Calls.

Last Modified: 11/19/2015 03:22am

C#

C# SOAP v4_1 Examples.

Last Modified: 11/19/2015 03:52am

Creating or Updating a Record

Overview

A C# example demonstrating how to create or update an account with the set_entry method using SOAP and the v4 SOAP API.

Example

```
using System;
using System.Text;
using System.Security.Cryptography;
using System.Collections.Specialized;

namespace SugarSoap
{
    class Program
    {
        static void Main(string[] args)
        {
            //login ----------------------------------------

            string UserName = "admin";
            string Password = "password";
            string URL = "http://{site_url}/service/v4/soap.php";

            //SugarCRM is a web reference added that points to
http://{site_url}/service/v4/soap.php?wsdl
            SugarCRM.sugarsoap SugarClient = new SugarCRM.sugarsoap();
            SugarClient.Timeout = 900000;
            SugarClient.Url = URL;

            string SessionID = String.Empty;

            //Create authentication object
            SugarCRM.user_auth UserAuth = new SugarCRM.user_auth();

            //Populate credentials
            UserAuth.user_name = UserName;
            UserAuth.password = getMD5(Password);

            //Try to authenticate
            SugarCRM.name_value[] LoginList = new
SugarCRM.name_value[0];
            SugarCRM.entry_value LoginResult =
SugarClient.login(UserAuth, "SoapTest", LoginList);

            //get session id
            SessionID = LoginResult.id;
```

```
            //create account -------------------------------

            NameValueCollection fieldListCollection = new
NameValueCollection();
            //to update a record, you will nee to pass in a record id
as commented below
            //fieldListCollection.Add("id",
"68c4781f-75d1-223a-5d8f-5058bc4e39ea");
            fieldListCollection.Add("name", "Test Account");

            //this is just a trick to avoid having to manually specify
index values for name_value[]
            SugarCRM.name_value[] fieldList = new
SugarCRM.name_value[fieldListCollection.Count];

            int count = 0;
            foreach (string name in fieldListCollection)
            {
                foreach (string value in
fieldListCollection.GetValues(name))
                {
                    SugarCRM.name_value field = new
SugarCRM.name_value();
                    field.name = name; field.value = value;
                    fieldList[count] = field;
                }
                count++;
            }

            try
            {
                SugarCRM.new_set_entry_result result =
SugarClient.set_entry(SessionID, "Accounts", fieldList);
                string RecordID = result.id;

                //show record id to user
                Console.WriteLine(RecordID);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.Source);
            }

            //Pause Window
```

```
            Console.ReadLine();
        }

        static private string getMD5(string PlainText)
        {
            MD5 md5 = MD5.Create();
            byte[] inputBuffer =
System.Text.Encoding.ASCII.GetBytes(PlainText);
            byte[] outputBuffer = md5.ComputeHash(inputBuffer);

            //Convert the byte[] to a hex-string
            StringBuilder builder = new
StringBuilder(outputBuffer.Length);
            for (int i = 0; i < outputBuffer.Length; i++)
            {
                builder.Append(outputBuffer[i].ToString("X2"));
            }

            return builder.ToString();
        }
    }
}
```

Result

```
68c4781f-75d1-223a-5d8f-5058bc4e39ea
```

Last Modified: 09/26/2015 04:23pm

Logging In

Overview

A C# example demonstrating how to log in and retrieve a session key using SOAP and the v4 SOAP API.

Example

```
using System;
using System.Text;
```

```
using System.Security.Cryptography;

namespace SugarSoap
{
    class Program
    {
        static void Main(string[] args)
        {
            string UserName = "admin";
            string Password = "password";
            string URL = "http://{site_url}/service/v4/soap.php";
            string SessionID = String.Empty;

            //SugarCRM is a web reference added that points to
http://{site_url}/service/v4/soap.php?wsdl
            SugarCRM.sugarsoap SugarClient = new SugarCRM.sugarsoap();
            SugarClient.Timeout = 900000;
            SugarClient.Url = URL;

            //Create authentication object
            SugarCRM.user_auth UserAuth = new SugarCRM.user_auth();

            //Populate credentials
            UserAuth.user_name = UserName;
            UserAuth.password = getMD5(Password);

            //Try to authenticate
            SugarCRM.name_value[] LoginList = new
SugarCRM.name_value[0];
            SugarCRM.entry_value LoginResult =
SugarClient.login(UserAuth, "SoapTest", LoginList);

            //get session id
            SessionID = LoginResult.id;

            if (SessionID != String.Empty)
            {
                //print session
                Console.WriteLine(SessionID);
            }

            //Pause Window
            Console.ReadLine();
        }

        static private string getMD5(string TextString)
```

```
        {
            MD5 md5 = MD5.Create();
            byte[] inputBuffer =
System.Text.Encoding.ASCII.GetBytes(TextString);
            byte[] outputBuffer = md5.ComputeHash(inputBuffer);

            StringBuilder Builder = new
StringBuilder(outputBuffer.Length);
            for (int i = 0; i < outputBuffer.Length; i++)
            {
                Builder.Append(outputBuffer[i].ToString("X2"));
            }

            return Builder.ToString();
        }
    }
}
```

Result

```
b2uv0vrjfiov41d03sk578ufq6
```

Last Modified: 09/26/2015 04:23pm

PHP

PHP SOAP v4_1 Examples.

Last Modified: 11/19/2015 03:52am

Creating Documents

Overview

A PHP example demonstrating how to create a document using set_entry and a document revision with the set_document_revision method using NuSOAP and the v4_1 SOAP API.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
    $username = "admin";
    $password = "password";

    //require NuSOAP
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');

    //display errors
    $err = $client->getError();
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login ----------------------------------------------------
    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => md5($password),
              'version' => '1'
         ),
         'application_name' => 'SoapTest',
         'name_value_list' => array(
         ),
    );

    $login_result = $client->call('login', $login_parameters);

    /*
    echo '<pre>';
    print_r($login_result);
    echo '</pre>';
    */

    //get session id
    $session_id =  $login_result['id'];

    //create document ------------------------------------------
```

```
    $set_entry_parameters = array(
        //session id
        "session" => $session_id,

        //The name of the module
        "module_name" => "Documents",

        //Record attributes
        "name_value_list" => array(
            //to update a record, you will nee to pass in a record id
as commented below
            //array("name" => "id", "value" =>
"9b170af9-3080-e22b-fbc1-4fea74def88f"),
            array("name" => "document_name", "value" => "Example
Document"),
            array("name" => "revision", "value" => "1"),
        ),
    );

    $set_entry_result = $client->call("set_entry",
$set_entry_parameters);

    echo "<pre>";
    print_r($set_entry_result);
    echo "</pre>";

    $document_id = $set_entry_result['id'];

    //create document revision ------------------------------------
    $contents = file_get_contents ("/path/to/example_document.txt");

    $set_document_revision_parameters = array(
        //session id
        "session" => $session_id,

        //The attachment details
        "note" => array(
            //The ID of the parent document.
            'id' => $document_id,

            //The binary contents of the file.
            'file' => base64_encode($contents),

            //The name of the file
            'filename' => 'example_document.txt',
```

```
            //The revision number
            'revision' => '1',
        ),
    );

    $set_document_revision_result =
$client->call("set_document_revision",
$set_document_revision_parameters);

    echo "<pre>";
    print_r($set_document_revision_result);
    echo "</pre>";

?>
```

Result

```
//set_entry result
Array
(
    [id] => 5ab53b9d-2f31-9b69-d92b-50abc2d0f6a2
)

//set_document_revision result
Array
(
    [id] => 906ad157-0aa0-c01e-2694-50abc2adcbf2
)
```

Last Modified: 09/26/2015 04:23pm

Creating Notes with Attachments

Overview

A PHP example demonstrating how to create a note using set_entry and an attachment with the set_note_attachment method using NuSOAP and the v4_1 SOAP API.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
    $username = "admin";
    $password = "password";

    //require NuSOAP
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');

    //display errors
    $err = $client->getError();
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login ----------------------------------------------------
    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => md5($password),
              'version' => '1'
         ),
         'application_name' => 'SoapTest',
         'name_value_list' => array(
         ),
    );

    $login_result = $client->call('login', $login_parameters);

    /*
    echo '<pre>';
    print_r($login_result);
    echo '</pre>';
    */

    //get session id
    $session_id =  $login_result['id'];

    //create note ----------------------------------------------
```

```
    $set_entry_parameters = array(
        //session id
        "session" => $session_id,

        //The name of the module
        "module_name" => "Notes",

        //Record attributes
        "name_value_list" => array(
            //to update a record, you will nee to pass in a record id
as commented below
            //array("name" => "id", "value" =>
"9b170af9-3080-e22b-fbc1-4fea74def88f"),
            array("name" => "name", "value" => "Example Note"),
        ),
    );

    $set_entry_result = $client->call("set_entry",
$set_entry_parameters);

    echo "<pre>";
    print_r($set_entry_result);
    echo "</pre>";

    $note_id = $set_entry_result['id'];

    //create note attachment ------------------------------------
    $contents = file_get_contents ("/path/to/example_file.php");

    $set_note_attachment_parameters = array(
        //session id
        "session" => $session_id,

        //The attachment details
        "note" => array(
            //The ID of the note containing the attachment.
            'id' => $note_id,

            //The file name of the attachment.
            'filename' => 'example_file.php',

            //The binary contents of the file.
            'file' => base64_encode($contents),
        ),
    );
```

```
    $set_note_attachment_result = $client->call("set_note_attachment",
$set_note_attachment_parameters);

    echo "<pre>";
    print_r($set_note_attachment_result);
    echo "</pre>";

?>
```

Result

```
//set_entry_result
Array
(
    [id] => 59a33435-7c6a-2d97-e61b-50abcc5d2644
)

//set_note_attachment result
Array
(
    [id] => 59a33435-7c6a-2d97-e61b-50abcc5d2644
)
```

Last Modified: 09/26/2015 04:23pm

Creating or Updating a Record

Overview

A PHP example demonstrating how to create or update an Account with the set_entry method using NuSOAP and the v4_1 SOAP API.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
    $username = "admin";
    $password = "password";

    //require NuSOAP
```

```
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');

    //display errors
    $err = $client->getError();
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login ----------------------------------------------------
    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => md5($password),
              'version' => '1'
         ),
         'application_name' => 'SoapTest',
         'name_value_list' => array(
         ),
    );

    $login_result = $client->call('login', $login_parameters);

    /*
    echo '<pre>';
    print_r($login_result);
    echo '</pre>';
    */

    //get session id
    $session_id =  $login_result['id'];

    //create account -------------------------------------
    $set_entry_parameters = array(
         //session id
         "session" => $session_id,

         //The name of the module from which to retrieve records.
         "module_name" => "Accounts",
```

```
        //Record attributes
        "name_value_list" => array(
             //to update a record, you will nee to pass in a record
id as commented below
             //array("name" => "id", "value" =>
"9b170af9-3080-e22b-fbc1-4fea74def88f"),
             array("name" => "name", "value" => "Test Account"),
        ),
    );

    $set_entry_result = $client->call("set_entry",
$set_entry_parameters);

    echo "<pre>";
    print_r($set_entry_result);
    echo "</pre>";

?>
```

Result

```
Array
(
    [id] => 63c103dd-1f47-804c-1282-52af64b870d4
)
```

Last Modified: 09/26/2015 04:23pm

Creating or Updating Multiple Records

Overview

A PHP example demonstrating how to create or update multiple contacts with the set_entries method using NuSOAP and the v4_1 SOAP API.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
    $username = "admin";
```

```
    $password = "password";

    //require NuSOAP
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');

    //display errors
    $err = $client->getError();
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login ----------------------------------------------------
    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => md5($password),
              'version' => '1'
         ),
         'application_name' => 'SoapTest',
         'name_value_list' => array(
         ),
    );

    $login_result = $client->call('login', $login_parameters);

    /*
    echo '<pre>';
    print_r($login_result);
    echo '</pre>';
    */

    //get session id
    $session_id =  $login_result['id'];

    //create contacts ------------------------------------
    $set_entries_parameters = array(
         //session id
         "session" => $session_id,
```

```
        //The name of the module from which to retrieve records.
        "module_name" => "Contacts",

        //Record attributes
        "name_value_list" => array(
            array(
                //to update a record, you will nee to pass in a record
id as commented below
                //array("name" => "id", "value" =>
"912e58c0-73e9-9cb6-c84e-4ff34d62620e"),
                array("name" => "first_name", "value" => "John"),
                array("name" => "last_name", "value" => "Smith"),
            ),
            array(
                //to update a record, you will nee to pass in a record
id as commented below
                //array("name" => "id", "value" =>
"99d6ddfd-7d52-d45b-eba8-4ff34d684964"),
                array("name" => "first_name", "value" => "Jane"),
                array("name" => "last_name", "value" => "Doe"),
            ),
        ),
    );

    $set_entries_result = $client->call("set_entries",
$set_entries_parameters);
    echo "<pre>";
    print_r($set_entries_result);
    echo "</pre>";

?>
```

Result

```
Array
(
    [ids] => Array
        (
            [0] => 912e58c0-73e9-9cb6-c84e-4ff34d62620e
            [1] => 99d6ddfd-7d52-d45b-eba8-4ff34d684964
        )
)
```

Last Modified: 09/26/2015 04:23pm

Creating or Updating Teams

Overview

A PHP example demonstrating how to manipulate teams using NuSOAP and the v4_1 SOAP API.

Note: If you are creating a private team for a user, you will need to set private to true and populate the associated_user_id populated. You should also populate the name and name_2 properties with the users first and last name.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
    $username = "admin";
    $password = "password";

    //require NuSOAP
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');

    //display errors
    $err = $client->getError();
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login ----------------------------------------------------
    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => md5($password),
              'version' => '1'
         ),
```

```
        'application_name' => 'SoapTest',
        'name_value_list' => array(
        ),
    );

    $login_result = $client->call('login', $login_parameters);

    /*
    echo '<pre>';
    print_r($login_result);
    echo '</pre>';
    */

    //get session id
    $session_id =  $login_result['id'];

    //create team -------------------------------------------
    $set_entry_parameters = array(

        // session id
        "session" => $session_id,

        // The name of the module that the record will be create in.
        "module_name" => "Teams",

        // array of arrays for the record attributes
        "name_value_list" => array(
            /* Setting the id with a valid record id will update the
record.
            array(
                "name" => "id",
                "value" => "47dbab1d-bd78-09e8-4392-5256b4501d90"
            ),
            */
            array(
                "name" => "name",
                "value" => "My Team"
            ),
            array(
                "name" => "description",
                "value" => "My new team"
            ),
            //Whether the team is private.
            //Private teams will have the associated_user_id
populated.
            array(
```

```
                "name" => "private",
                "value" => 0
            ),
        ),
    );

    $set_entry_result = $client->call('set_entry',
$set_entry_parameters);

    echo "<pre>";
    print_r($set_entry_result);
    echo "</pre>";

?>
```

Result

```
Array
(
    [id] => 90130b4f-4d39-100b-98de-52ab7a638d0d
)
```

Last Modified: 09/26/2015 04:23pm

Logging In

Overview

A PHP example demonstrating how to log in and retrieve a session key using NuSOAP and the v4_1 SOAP API.

Standard Authentication Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
    $username = "admin";
    $password = "password";

    //require NuSOAP
```

```
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');

    //display errors
    $err = $client->getError();
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login ----------------------------------------------------
    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => md5($password),
              'version' => '1'
         ),
         'application_name' => 'SoapTest',
         'name_value_list' => array(
         ),
    );

    $login_result = $client->call('login', $login_parameters);

    echo '<pre>';
    print_r($login_result);
    echo '</pre>';

    //get session id
    $session_id =  $login_result['id'];

?>
```

LDAP Authentication Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
    $username = "admin";
    $password = "password";
```

```
    $ldap_enc_key = 'LDAP_ENCRYPTION_KEY';

    //require NuSOAP
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');

    //display errors
    $err = $client->getError();
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login ----------------------------------------------------
    $ldap_enc_key = substr(md5($ldap_enc_key), 0, 24);
    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => bin2hex(mcrypt_cbc(MCRYPT_3DES,
$ldap_enc_key, $password, MCRYPT_ENCRYPT, 'password')),
              'version' => '1'
         ),

         'application_name' => 'SoapTest',
         'name_value_list' => array(
         ),
    );

    $login_result = $client->call('login', $login_parameters);

    echo '<pre>';
    print_r($login_result);
    echo '</pre>';

    //get session id
    $session_id =  $login_result['id'];

?>
```

Result

```
Array
(
    [id] => 9uukc92a6lb9v620reqv34mmg1
    [module_name] => Users
    [name_value_list] => Array
        (
            [0] => Array
                (
                    [name] => user_id
                    [value] => 1
                )

            [1] => Array
                (
                    [name] => user_name
                    [value] => admin
                )

            [2] => Array
                (
                    [name] => user_language
                    [value] => en_us
                )

            [3] => Array
                (
                    [name] => user_currency_id
                    [value] => -99
                )

            [4] => Array
                (
                    [name] => user_is_admin
                    [value] => 1
                )

            [5] => Array
                (
                    [name] => user_default_team_id
                    [value] => 1
                )

            [6] => Array
                (
```

```
                [name] => user_default_dateformat
                [value] => m/d/Y
            )

        [7] => Array
            (
                [name] => user_default_timeformat
                [value] => h:ia
            )

        [8] => Array
            (
                [name] => user_number_seperator
                [value] => ,
            )

        [9] => Array
            (
                [name] => user_decimal_seperator
                [value] => .
            )

        [10] => Array
            (
                [name] => mobile_max_list_entries
                [value] => 10
            )

        [11] => Array
            (
                [name] => mobile_max_subpanel_entries
                [value] => 3
            )

        [12] => Array
            (
                [name] => user_currency_name
                [value] => US Dollars
            )
    )
)
```

Last Modified: 09/26/2015 04:23pm

Relating Quotes and Products

Overview

A PHP example demonstrating how to create and relate Products to Quotes using and the v4_1 SOAP API.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
    $username = "admin";
    $password = "password";

    //require NuSOAP
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');

    //display errors
    $err = $client->getError();
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login ----------------------------------------------------

    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => md5($password),
              'version' => '1'
         ),
         'application_name' => 'SoapTest',
         'name_value_list' => array(
         ),
    );
```

```
    $login_result = $client->call("login", $login_parameters);

    /*
    echo "<pre>";
    print_r($login_result);
    echo "</pre>";
    */

    //get session id
    $session_id = $login_result['id'];

    //create quote ----------------------------------------------------
    $createQuoteParams = array(
        'session' => $session_id,
        'module_name' => 'Quotes',
        'name_value_list' => array(
            array(
                'name' => 'name',
                'value' => 'Widget Quote'
            ),
            array(
                'name' => 'team_count',
                'value' => ''
            ),
            array(
                'name' => 'team_name',
                'value' => ''
            ),
            array(
                'name' => 'date_quote_expected_closed',
                'value' => date('Y-m-d', mktime(0, 0, 0, date('m')  ,
date('d')+7, date('Y')))
            ),
            array(
                'name' => 'quote_stage',
                'value' => 'Negotiation'
            ),
            array(
                'name' => 'quote_num',
                'value' => ''
            ),
            array(
                'name' => 'quote_type',
                'value' => 'Quotes'
            ),
```

```
            array(
                'name' => 'subtotal',
                'value' => '1230.23'
            ),
            array(
                'name' => 'subtotal_usdollar',
                'value' => '1230.23'
            ),
        ),
    );

    $createQuoteResult = $client->call('set_entry',
$createQuoteParams);

    echo "Create Quote Result<br />";
    echo "<pre>";
    print_r($createQuoteResult);
    echo "</pre>";

    //create product ------------------------------------------------
    $createProductParams = array(
        'session' => $session_id,
        'module_name' => 'Products',
        'name_value_list' => array(
            array(
                'name' => 'name',
                'value' => 'Widget'
            ),
            array(
                'name' => 'quote_id',
                'value' => $createQuoteResult['id']
            ),
            array(
                'name' => 'status',
                'value' => 'Quotes'
            )
        )
    );

    $createProductResult = $client->call('set_entry',
$createProductParams);

    echo "Create Product Result<br />";
    echo "<pre>";
    print_r($createProductResult);
    echo "</pre>";
```

```
    //create product-bundle
-------------------------------------------------
    $createProductBundleParams = array(
        "session" => $session_id,
        "module_name" => "ProductBundles",
        "name_value_list" => array(
            array(
                'name' => 'name',
                'value' => 'Order'),
            array(
                'name' => 'bundle_stage',
                'value' => 'Draft'
            ),
            array(
                'name' => 'tax',
                'value' => '0.00'
            ),
            array(
                'name' => 'total',
                'value' => '0.00'
            ),
            array(
                'name' => 'subtotal',
                'value' => '0.00'
            ),
            array(
                'name' => 'shippint',
                'value' => '0.00'
            ),
            array(
                'name' => 'currency_id',
                'value' => '-99'
            ),
        )
    );

    $createProductBundleResult = $client->call('set_entry',
$createProductBundleParams);

    echo "Create ProductBundles Result<br />";
    echo "<pre>";
    print_r($createProductBundleResult);
    echo "</pre>";

    //relate product to product-bundle
```

```
---------------------------------------
    $relationshipProductBundleProductsParams = array(
        'session' => $session_id,
        'module_name' => 'ProductBundles',
        'module_id' => $createProductBundleResult['id'],
        'link_field_name' => 'products',
        'related_ids' => array(
            $createProductResult['id']
        ),
        'name_value_list' => array(),
        'delete' => 0
    );

    // set the product bundles products relationship
    $relationshipProductBundleProductResult =
$client->call('set_relationship',
$relationshipProductBundleProductsParams);

    echo "Create ProductBundleProduct Relationship Result<br />";
    echo "<pre>";
    print_r($relationshipProductBundleProductResult);
    echo "</pre>";

    //relate product-bundle to quote
---------------------------------------
    $relationshipProductBundleQuoteParams = array(
        'session' => $session_id,
        'module_name' => 'Quotes',
        'module_id' => $createQuoteResult['id'],
        'link_field_name' => 'product_bundles',
        'related_ids' => array(
            $createProductBundleResult['id']
        ),
        'name_value_list' => array(),
        'delete' => 0
    );

    // set the product bundles quotes relationship
    $relationshipProductBundleQuoteResult =
$client->call('set_relationship',
$relationshipProductBundleQuoteParams);

    echo "Create ProductBundleQuote Relationship Result<br />";
    echo "<pre>";
    print_r($relationshipProductBundleQuoteResult);
    echo "</pre>";
```

Result

```
//Create Quote Result
Array
(
    [id] => ad88195a-9d36-20b6-beb1-517ea2b9278d
)

//Create Product Result
Array
(
    [id] => 6f920cba-0fed-7172-9dc0-517ea2adb1d1
)

//Create ProductBundles Result
Array
(
    [id] => 4f397baa-5522-fa0e-2bcf-517ea24a9546
)

//Create ProductBundleProduct Relationship Result
Array
(
    [created] => 1
    [failed] => 0
    [deleted] => 0
)

//Create ProductBundleQuote Relationship Result
Array
(
    [created] => 1
    [failed] => 0
    [deleted] => 0
)
```

Last Modified: 09/26/2015 04:23pm

Retrieving a List of Fields From a Module

Overview

A PHP example demonstrating how to retrieve fields vardefs from the accounts module with the get_module_fields method using NuSOAP and the v4_1 SOAP API.

This example will only retrieve the vardefs for the 'id' and 'name' fields.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
    $username = "admin";
    $password = "password";

    //require NuSOAP
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');

    //display errors
    $err = $client->getError();
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login ----------------------------------------------------
    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => md5($password),
              'version' => '1'
         ),
         'application_name' => 'SoapTest',
         'name_value_list' => array(
         ),
    );

    $login_result = $client->call('login', $login_parameters);

    /*
    echo '<pre>';
```

```
    print_r($login_result);
    echo '</pre>';
    */

    //get session id
    $session_id =  $login_result['id'];

    //retrieve fields ----------------------------------
    $get_module_fields_parameters = array(

        //session id
        'session' => $session_id,

        //The name of the module from which to retrieve records
        'module_name' => 'Accounts',

        //Optional. Returns vardefs for the specified fields. An empty
array will return all fields.
        'fields' => array(
            'id',
            'name',
        ),
    );

    $get_module_fields_result = $client->call("get_module_fields",
$get_module_fields_parameters);
    echo "<pre>";
    print_r($get_module_fields_result);
    echo "</pre>";

?>
```

Result

```
Array
(
    [module_name] => Accounts
    [table_name] => accounts
    [module_fields] => Array
        (
            [0] => Array
                (
                    [name] => id
                    [type] => id
                    [group] =>
```

```
                    [label] => ID
                    [required] => 1
                    [options] => Array
                        (
                        )
                )

            [1] => Array
                (
                    [name] => name
                    [type] => name
                    [group] =>
                    [label] => Name:
                    [required] => 1
                    [options] => Array
                        (
                        )
                )
        )

    [link_fields] => Array
        (
        )
)
```

Last Modified: 09/26/2015 04:23pm

Retrieving a List of Records

Overview

A PHP example demonstrating how to retrieve a list of records from a module with the get_entry_list method using NuSOAP and the v4_1 SOAP API. This example will retrieve a list of leads.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
    $username = "admin";
    $password = "password";
```

```
    //require NuSOAP
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');

    //display errors
    $err = $client->getError();
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login ------------------------------------------------
    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => md5($password),
              'version' => '1'
         ),
         'application_name' => 'SoapTest',
         'name_value_list' => array(
         ),
    );

    $login_result = $client->call('login', $login_parameters);

    /*
    echo '<pre>';
    print_r($login_result);
    echo '</pre>';
    */

    //get session id
    $session_id =  $login_result['id'];

    //get list of records ----------------------------------
    $get_entry_list_parameters = array(

         //session id
         'session' => $session_id,
```

```
        //The name of the module from which to retrieve records
        'module_name' => 'Leads',

        //The SQL WHERE clause without the word "where".
        'query' => "",

        //The SQL ORDER BY clause without the phrase "order by".
        'order_by' => "",

        //The record offset from which to start.
        'offset' => '0',

        //Optional. A list of fields to include in the results.
        'select_fields' => array(
             'id',
             'name',
             'title',
        ),

        /*
        A list of link names and the fields to be returned for each
link name.
        Example: 'link_name_to_fields_array' => array(array('name' =>
'email_addresses', 'value' => array('id', 'email_address', 'opt_out',
'primary_address')))
        */
        'link_name_to_fields_array' => array(
        ),

        //The maximum number of results to return.
        'max_results' => '2',

        //To exclude deleted records
        'deleted' => '0',

        //If only records marked as favorites should be returned.
        'Favorites' => false,
    );

    $get_entry_list_result = $client->call('get_entry_list',
$get_entry_list_parameters);
    echo '<pre>';
    print_r($get_entry_list_result);
    echo '</pre>';

?>
```

Result

```
Array
(
    [result_count] => 2
    [total_count] => 200
    [next_offset] => 2
    [entry_list] => Array
        (
            [0] => Array
                (
                    [id] => 18124607-69d1-b158-47ff-4f7cb69344f7
                    [module_name] => Leads
                    [name_value_list] => Array
                        (
                            [0] => Array
                                (
                                    [name] => id
                                    [value] =>
18124607-69d1-b158-47ff-4f7cb69344f7
                                )

                            [1] => Array
                                (
                                    [name] => name
                                    [value] => Bernie Worthey
                                )

                            [2] => Array
                                (
                                    [name] => title
                                    [value] => Senior Product Manager
                                )
                        )
                )

            [1] => Array
                (
                    [id] => 1cdfddc1-2759-b007-8713-4f7cb64c2e9c
                    [module_name] => Leads
                    [name_value_list] => Array
                        (
                            [0] => Array
                                (
```

```
                                    [name] => id
                                    [value] =>
1cdfddc1-2759-b007-8713-4f7cb64c2e9c
                                )

                            [1] => Array
                                (
                                    [name] => name
                                    [value] => Bobbie Kohlmeier
                                )

                            [2] => Array
                                (
                                    [name] => title
                                    [value] => Director Operations
                                )
                        )
                )
        )

    [relationship_list] => Array
        (
        )
)
```

Last Modified: 09/26/2015 04:23pm

Retrieving a List of Records With Related Info

Overview

A PHP example demonstrating how to retrieve a list of records with info from a related entity with the get_entry_list method using NuSOAP and the v4_1 SOAP API.

This example will retrieve a list of contacts and their related email addresses.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
```

```
    $username = "admin";
    $password = "password";

    //require NuSOAP
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');

    //display errors
    $err = $client->getError();
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login ----------------------------------------------------
    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => md5($password),
              'version' => '1'
         ),
         'application_name' => 'SoapTest',
         'name_value_list' => array(
         ),
    );

    $login_result = $client->call('login', $login_parameters);

    /*
    echo '<pre>';
    print_r($login_result);
    echo '</pre>';
    */

    //get session id
    $session_id =  $login_result['id'];

    //retrieve records -----------------------------------
    $get_entry_list_parameters = array(

         //session id
```

```
        'session' => $session_id,

        //The name of the module from which to retrieve records
        'module_name' => "Contacts",

        //The SQL WHERE clause without the word "where".
        'query' => "",

        //The SQL ORDER BY clause without the phrase "order by".
        'order_by' => "",

        //The record offset from which to start.
        'offset' => "0",

        //Optional. The list of fields to be returned in the results
        'select_fields' => array(
             'id',
             'first_name',
             'last_name',
        ),

        //A list of link names and the fields to be returned for each
link name
        'link_name_to_fields_array' => array(
             array(
                  'name' => 'email_addresses',
                  'value' => array(
                       'id',
                       'email_address',
                       'opt_out',
                       'primary_address'
                  ),
             ),
        ),

        //The maximum number of results to return.
        'max_results' => '2',

        //To exclude deleted records
        'deleted' => 0,

        //If only records marked as favorites should be returned.
        'Favorites' => false,
    );

    $get_entry_list_result = $client->call("get_entry_list",
```

```
$get_entry_list_parameters);
    echo "<pre>";
    print_r($get_entry_list_result);
    echo "</pre>";

?>
```

Result

```
Array
(
   [result_count] => 2
   [total_count] => -1
   [next_offset] => 2
   [entry_list] => Array
   (
      [0] => Array
      (
         [id] => 10613769-54e0-820d-de9b-5282a31b88cd
         [module_name] => Contacts
         [name_value_list] => Array
         (
            [0] => Array
            (
               [name] => id
               [value] => 10613769-54e0-820d-de9b-5282a31b88cd
            )

            [1] => Array
            (
               [name] => first_name
               [value] => Son
            )

            [2] => Array
            (
               [name] => last_name
               [value] => Roan
            )
         )
      )

      [1] => Array
      (
         [id] => 129b5f1d-5f1f-c679-beab-5282a325a501
```

```
        [module_name] => Contacts
        [name_value_list] => Array
        (
           [0] => Array
           (
              [name] => id
              [value] => 129b5f1d-5f1f-c679-beab-5282a325a501
           )

           [1] => Array
           (
              [name] => first_name
              [value] => Pasquale
           )

           [2] => Array
           (
              [name] => last_name
              [value] => Gottlieb
           )
        )
     )
  )

  [relationship_list] => Array
  (
     [0] => Array
     (
        [link_list] => Array
        (
           [0] => Array
           (
              [name] => email_addresses
              [records] => Array
              (
                 [0] => Array
                 (
                    [link_value] => Array
                    (
                       [0] => Array
                       (
                          [name] => id
                          [value] =>
11980e8f-9cb6-0019-ad4a-5282a3e705cf
                       )
```

```
                        [1] => Array
                        (
                           [name] => email_address
                           [value] => kid76@example.us
                        )

                        [2] => Array
                        (
                           [name] => opt_out
                           [value] => 0
                        )

                        [3] => Array
                        (
                           [name] => primary_address
                           [value] =>
                        )
                     )
                  )

                  [1] => Array
                  (
                     [link_value] => Array
                     (
                        [0] => Array
                        (
                           [name] => id
                           [value] =>
11c82450-4664-b210-4b79-5282a339d730
                        )

                        [1] => Array
                        (
                           [name] => email_address
                           [value] => info.dev@example.name
                        )

                        [2] => Array
                        (
                           [name] => opt_out
                           [value] => 1
                        )

                        [3] => Array
                        (
                           [name] => primary_address
```

```
                        [value] =>
                    )
                )
            )
        )
    )
)

[1] => Array
(
    [link_list] => Array
    (
        [0] => Array
        (
            [name] => email_addresses
            [records] => Array
            (
                [0] => Array
                (
                    [link_value] => Array
                    (
                        [0] => Array
                        (
                            [name] => id
                            [value] =>
13bf6b4b-4feb-823f-7c54-5282a36a2f57
                        )

                        [1] => Array
                        (
                            [name] => email_address
                            [value] => phone.vegan@example.tv
                        )

                        [2] => Array
                        (
                            [name] => opt_out
                            [value] => 0
                        )

                        [3] => Array
                        (
                            [name] => primary_address
                            [value] =>
                        )
```

```
                    )
                )

                [1] => Array
                (
                    [link_value] => Array
                    (
                        [0] => Array
                        (
                            [name] => id
                            [value] =>
13ee5142-17c1-76fd-3cc4-5282a32cf864
                        )

                        [1] => Array
                        (
                            [name] => email_address
                            [value] => kid.hr.sugar@example.biz
                        )

                        [2] => Array
                        (
                            [name] => opt_out
                            [value] => 1
                        )

                        [3] => Array
                        (
                            [name] => primary_address
                            [value] =>
                        )
                    )
                )
            )
        )
    )
)
```

Last Modified: 09/26/2015 04:23pm

Retrieving Multiple Records by ID

Overview

A PHP example demonstrating how to retrieve multiple records from the accounts module with the get_entries method using NuSOAP and the v4_1 SOAP API.

This example will only retrieve 2 records and return the following fields: 'name', 'billing_address_state' and 'billing_address_country'.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
    $username = "admin";
    $password = "password";

    //require NuSOAP
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');

    //display errors
    $err = $client->getError();
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login
----------------------------------------------------------------
    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => md5($password),
              'version' => '1'
         ),
         'application_name' => 'SoapTest',
         'name_value_list' => array(
         ),
    );
```

```
    $login_result = $client->call('login', $login_parameters);

    /*
    echo '<pre>';
    print_r($login_result);
    echo '</pre>';
    */

    //get session id
    $session_id =  $login_result['id'];

    //retrieve records
-------------------------------------------------
    $get_entries_parameters = array(

         //session id
         'session' => $session_id,

         //The name of the module from which to retrieve records
         'module_name' => 'Accounts',

         //An array of SugarBean IDs
         'ids' => array(
             '20328809-9d0a-56fc-0e7c-4f7cb6eb1c83',
             '328b22a6-d784-66d9-0295-4f7cb59e8cbb',
         ),

         //Optional. The list of fields to be returned in the results
         'select_fields' => array(
            'name',
            'billing_address_state',
            'billing_address_country'
         ),

         //A list of link names and the fields to be returned for each
link name
         'link_name_to_fields_array' => array(
         ),
    );

    $get_entries_result = $client->call('get_entries',
$get_entries_parameters);

    echo '<pre>';
    print_r($get_entries_result);
    echo '</pre>';
```

```
?>
```

Result

```
Array
(
    [entry_list] => Array
        (
            [0] => Array
                (
                    [id] => 20328809-9d0a-56fc-0e7c-4f7cb6eb1c83
                    [module_name] => Accounts
                    [name_value_list] => Array
                        (
                            [0] => Array
                                (
                                    [name] => name
                                    [value] => Jungle Systems Inc
                                )

                            [1] => Array
                                (
                                    [name] => billing_address_state
                                    [value] => NY
                                )

                            [2] => Array
                                (
                                    [name] => billing_address_country
                                    [value] => USA
                                )
                        )
                )

            [1] => Array
                (
                    [id] => 328b22a6-d784-66d9-0295-4f7cb59e8cbb
                    [module_name] => Accounts
                    [name_value_list] => Array
                        (
                            [0] => Array
                                (
                                    [name] => name
                                    [value] => Riviera Hotels
```

```
                                )

                            [1] => Array
                                (
                                    [name] => billing_address_state
                                    [value] => CA
                                )

                            [2] => Array
                                (
                                    [name] => billing_address_country
                                    [value] => USA
                                )
                        )
                )
        )

    [relationship_list] => Array
        (
        )
)
```

Last Modified: 09/26/2015 04:23pm

Retrieving Records by Email Domain

Overview

A PHP example demonstrating how to retrieve email addresses based on an email domain with the search_by_module and get_entries methods using NuSOAP and the v4_1 SOAP API.

When using the search_by_module method, the email address information is not returned in the result. Due to this behavior, we will gather the record ids and pass them back to the get_entries method to fetch our related email addresses.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
    $username = "admin";
```

```
    $password = "password";

    //require NuSOAP
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');

    //display errors
    $err = $client->getError();
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login
----------------------------------------------------------------
    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => md5($password),
              'version' => '1'
         ),
         'application_name' => 'SoapTest',
         'name_value_list' => array(
         ),
    );

    $login_result = $client->call('login', $login_parameters);

    /*
    echo '<pre>';
    print_r($login_result);
    echo '</pre>';
    */

    //get session id
    $session_id =  $login_result['id'];

    //search_by_module ------------------------------------------
    $search_by_module_parameters = array(
        "session" => $session_id,
        'search_string' => '%@example.com',
```

```
        'modules' => array(
            'Accounts',
            'Contacts',
            'Leads',
        ),
        'offset' => 0,
        'max_results' => 1,
        'assigned_user_id' => '',
        'select_fields' => array('id'),
        'unified_search_only' => false,
        'favorites' => false
    );

    $search_by_module_results = $client->call('search_by_module',
$search_by_module_parameters);

    /*
    echo '<pre>';
    print_r($search_by_module_results);
    echo '</pre>';
    */

    $record_ids = array();
    foreach ($search_by_module_results['entry_list'] as $results)
    {
        $module = $results['name'];

        foreach ($results['records'] as $records)
        {
            foreach($records as $record)
            {
                if ($record['name'] = 'id')
                {
                    $record_ids[$module][] = $record['value'];
                    //skip any additional fields
                    break;
                }
            }
        }

    }

    $get_entries_results = array();
    $modules = array_keys($record_ids);

    foreach($modules as $module)
```

```
    {
        $get_entries_parameters = array(
            //session id
            'session' => $session_id,

            //The name of the module from which to retrieve records
            'module_name' => $module,

            //An array of record IDs
            'ids' => $record_ids[$module],

            //The list of fields to be returned in the results
            'select_fields' => array(
                'name',
            ),

            //A list of link names and the fields to be returned for
each link name
            'link_name_to_fields_array' => array(
                array(
                    'name' => 'email_addresses',
                    'value' => array(
                        'email_address',
                        'opt_out',
                        'primary_address'
                    ),
                ),
            ),

            //Flag the record as a recently viewed item
            'track_view' => false,
        );

        $get_entries_results[$module] = $client->call('get_entries',
$get_entries_parameters);
    }

    echo '<pre>';
    print_r($get_entries_results);
    echo '</pre>';
```

Result

```
Array
```

```
(
   [Accounts] => Array
   (
      [entry_list] => Array
      (
         [0] => Array
         (
            [id] => 1bb7ef28-64b9-cbd5-e7d6-5282a3b96953
            [module_name] => Accounts
            [name_value_list] => Array
            (
               [0] => Array
               (
                  [name] => name
                  [value] => Underwater Mining Inc.
               )
            )
         )

         [1] => Array
         (
            [id] => efbc0c4e-cc72-f843-b6ed-5282a38dad7f
            [module_name] => Accounts
            [name_value_list] => Array
            (
               [0] => Array
               (
                  [name] => name
                  [value] => 360 Vacations
               )
            )
         )
      )

      [relationship_list] => Array
      (
         [0] => Array
         (
            [link_list] => Array
            (
               [0] => Array
               (
                  [name] => email_addresses
                  [records] => Array
                  (
                     [0] => Array
```

```
(
    [link_value] => Array
    (
        [0] => Array
        (
            [name] => email_address
            [value] => beans.the.qa@example.com
        )

        [1] => Array
        (
            [name] => opt_out
            [value] => 0
        )

        [2] => Array
        (
            [name] => primary_address
            [value] =>
        )
    )
)

[1] => Array
(
    [link_value] => Array
    (
        [0] => Array
        (
            [name] => email_address
            [value] => section.sales@example.edu
        )

        [1] => Array
        (
            [name] => opt_out
            [value] => 0
        )

        [2] => Array
        (
            [name] => primary_address
            [value] =>
        )
    )
)
```

```
            )
        )
    )
)

[1] => Array
(
    [link_list] => Array
    (
        [0] => Array
        (
            [name] => email_addresses
            [records] => Array
            (
                [0] => Array
                (
                    [link_value] => Array
                    (
                        [0] => Array
                        (
                            [name] => email_address
                            [value] => section51@example.com
                        )

                        [1] => Array
                        (
                            [name] => opt_out
                            [value] => 0
                        )

                        [2] => Array
                        (
                            [name] => primary_address
                            [value] =>
                        )
                    )
                )

                [1] => Array
                (
                    [link_value] => Array
                    (
                        [0] => Array
                        (
                            [name] => email_address
                            [value] => kid.sugar.qa@example.co.uk
```

```
                        )

                        [1] => Array
                        (
                           [name] => opt_out
                           [value] => 0
                        )

                        [2] => Array
                        (
                           [name] => primary_address
                           [value] =>
                        )
                     )
                  )
               )
            )
         )
      )
   )
)

[Contacts] => Array
(
   [entry_list] => Array
   (
      [0] => Array
      (
         [id] => 24330979-0e39-f9ec-2622-5282a372f39b
         [module_name] => Contacts
         [name_value_list] => Array
         (
            [0] => Array
            (
               [name] => name
               [value] => Errol Goldberg
            )
         )
      )

      [1] => Array
      (
         [id] => 2542dad2-85f3-5529-3a30-5282a3104e31
         [module_name] => Contacts
         [name_value_list] => Array
         (
```

```
            [0] => Array
            (
               [name] => name
               [value] => Luis Deegan
            )
         )
      )
   )

   [relationship_list] => Array
   (
      [0] => Array
      (
         [link_list] => Array
         (
            [0] => Array
            (
               [name] => email_addresses
               [records] => Array
               (
                  [0] => Array
                  (
                     [link_value] => Array
                     (
                        [0] => Array
                        (
                           [name] => email_address
                           [value] => hr.phone@example.com
                        )

                        [1] => Array
                        (
                           [name] => opt_out
                           [value] => 0
                        )

                        [2] => Array
                        (
                           [name] => primary_address
                           [value] =>
                        )
                     )
                  )

                  [1] => Array
                  (
```

```
                [link_value] => Array
                (
                    [0] => Array
                    (
                        [name] => email_address
                        [value] => the.info.phone@example.cn
                    )

                    [1] => Array
                    (
                        [name] => opt_out
                        [value] => 1
                    )

                    [2] => Array
                    (
                        [name] => primary_address
                        [value] =>
                    )
                )
            )
        )
    )
)

[1] => Array
(
    [link_list] => Array
    (
        [0] => Array
        (
            [name] => email_addresses
            [records] => Array
            (
                [0] => Array
                (
                    [link_value] => Array
                    (
                        [0] => Array
                        (
                            [name] => email_address
                            [value] => im.the.sales@example.name
                        )

                        [1] => Array
```

```
                        (
                            [name] => opt_out
                            [value] => 0
                        )

                        [2] => Array
                        (
                            [name] => primary_address
                            [value] =>
                        )
                    )
                )

                [1] => Array
                (
                    [link_value] => Array
                    (
                        [0] => Array
                        (
                            [name] => email_address
                            [value] => the36@example.com
                        )

                        [1] => Array
                        (
                            [name] => opt_out
                            [value] => 1
                        )

                        [2] => Array
                        (
                            [name] => primary_address
                            [value] =>
                        )
                    )
                )
            )
        )
    )
)

[Leads] => Array
(
    [entry_list] => Array
```

```
(
    [0] => Array
    (
        [id] => 83abeeff-5e71-0f06-2e5f-5282a3f04f41
        [module_name] => Leads
        [name_value_list] => Array
        (
            [0] => Array
            (
                [name] => name
                [value] => Caryn Wert
            )
        )
    )

    [1] => Array
    (
        [id] => 2a3b304b-5167-8432-d4e7-5282a300eabb
        [module_name] => Leads
        [name_value_list] => Array
        (
            [0] => Array
            (
                [name] => name
                [value] => Marco Castonguay
            )
        )
    )
)

[relationship_list] => Array
(
    [0] => Array
    (
        [link_list] => Array
        (
            [0] => Array
            (
                [name] => email_addresses
                [records] => Array
                (
                    [0] => Array
                    (
                        [link_value] => Array
                        (
                            [0] => Array
```

```
                    (
                        [name] => email_address
                        [value] => hr60@example.com
                    )

                    [1] => Array
                    (
                        [name] => opt_out
                        [value] => 0
                    )

                    [2] => Array
                    (
                        [name] => primary_address
                        [value] =>
                    )
                )
            )
        )
    )
)

[1] => Array
(
    [link_list] => Array
    (
        [0] => Array
        (
            [name] => email_addresses
            [records] => Array
            (
                [0] => Array
                (
                    [link_value] => Array
                    (
                        [0] => Array
                        (
                            [name] => email_address
                            [value] => im.the@example.com
                        )

                        [1] => Array
                        (
                            [name] => opt_out
                            [value] => 0
```

```
                            )

                            [2] => Array
                            (
                               [name] => primary_address
                               [value] =>
                            )
                         )
                      )
                   )
                )
             )
          )
       )
    )
)
```

Last Modified: 09/26/2015 04:23pm

Retrieving Related Records

Overview

A PHP example demonstrating how to retrieve a list of related records with the get_relationships method using NuSOAP and the v4_1 SOAP API.

This example will retrieve a list of leads related to a specific target list.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
    $username = "admin";
    $password = "password";

    //require NuSOAP
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');
```

```
    //display errors
    $err = $client->getError();
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login ----------------------------------------------------
    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => md5($password),
              'version' => '1'
         ),
         'application_name' => 'SoapTest',
         'name_value_list' => array(
         ),
    );

    $login_result = $client->call('login', $login_parameters);

    /*
    echo '<pre>';
    print_r($login_result);
    echo '</pre>';
    */

    //get session id
    $session_id =  $login_result['id'];

    //retrieve related list ------------------------------------
    $get_relationships_parameters = array(

         'session'=>$session_id,

         //The name of the module from which to retrieve records.
         'module_name' => 'ProspectLists',

         //The ID of the specified module bean.
         'module_id' => '76d0e694-ef66-ddd5-9bdf-4febd3af44d5',

         //The relationship name of the linked field from which to
return records.
```

```
        'link_field_name' => 'leads',

        //The portion of the WHERE clause from the SQL statement used
to find the related items.
        'related_module_query' => '',

        //The related fields to be returned.
        'related_fields' => array(
           'id',
           'first_name',
           'last_name',
        ),

        //For every related bean returned, specify link field names
to field information.
        'related_module_link_name _to_fields_array' => array(
        ),

        //To exclude deleted records
        'deleted'=> '0',

        //order by
        'order_by' => '',

        //offset
        'offset' => 0,

        //limit
        'limit' => 5,
    );

    $get_relationships_result = $client->call("get_relationships",
$get_relationships_parameters);
    echo "<pre>";
    print_r($get_relationships_result);
    echo "</pre>";

?>
```

Result

```
Array
(
    [entry_list] => Array
        (
```

```
            [0] => Array
                (
                    [id] => 117c26c0-11d4-7b8b-cb8f-4f7cb6823dd8
                    [module_name] => Leads
                    [name_value_list] => Array
                        (
                            [0] => Array
                                (
                                    [name] => id
                                    [value] =>
117c26c0-11d4-7b8b-cb8f-4f7cb6823dd8
                                )

                            [1] => Array
                                (
                                    [name] => first_name
                                    [value] => Diane
                                )

                            [2] => Array
                                (
                                    [name] => last_name
                                    [value] => Mckamey
                                )
                        )
                )

            [1] => Array
                (
                    [id] => 142faeef-1a19-b53a-b780-4f7cb600c553
                    [module_name] => Leads
                    [name_value_list] => Array
                        (
                            [0] => Array
                                (
                                    [name] => id
                                    [value] =>
142faeef-1a19-b53a-b780-4f7cb600c553
                                )

                            [1] => Array
                                (
                                    [name] => first_name
                                    [value] => Leonor
                                )
```

```
                            [2] => Array
                                (
                                    [name] => last_name
                                    [value] => Reser
                                )
                        )
                )
        )

    [relationship_list] => Array
        (
        )
)
```

Last Modified: 09/26/2015 04:23pm

Searching Records

Overview

A PHP example demonstrating how to search the accounts module with the search_by_module method using NuSOAP and the v4_1 SOAP API.

This script will return two results, sorted by the id field, and return the value of the id, name, account_type, phone_office, and assigned_user_name fields.

Example

```
<?php

    $url = "http://{site_url}/service/v4_1/soap.php?wsdl";
    $username = "admin";
    $password = "password";

    //require NuSOAP
    require_once("./nusoap/lib/nusoap.php");

    //retrieve WSDL
    $client = new nusoap_client($url, 'wsdl');

    //display errors
    $err = $client->getError();
```

```
    if ($err)
    {
        echo '<h2>Constructor error</h2><pre>' . $err . '</pre>';
        echo '<h2>Debug</h2><pre>' .
htmlspecialchars($client->getDebug(), ENT_QUOTES) . '</pre>';
        exit();
    }

    //login ------------------------------------------------
    $login_parameters = array(
         'user_auth' => array(
              'user_name' => $username,
              'password' => md5($password),
              'version' => '1'
         ),
         'application_name' => 'SoapTest',
         'name_value_list' => array(
         ),
    );

    $login_result = $client->call('login', $login_parameters);

    /*
    echo '<pre>';
    print_r($login_result);
    echo '</pre>';
    */

    //get session id
    $session_id =  $login_result['id'];

    //search ----------------------------------------
    $search_by_module_parameters = array(
        //Session id
        "session" => $session_id,

        //The string to search for.
        'search_string' => 'Customer',

        //The list of modules to query.
        'modules' => array(
        'Accounts',
        ),

        //The record offset from which to start.
        'offset' => 0,
```

```
        //The maximum number of records to return.
        'max_results' => 2,

        //Filters records by the assigned user ID.
        //Leave this empty if no filter should be applied.
        'id' => '',

        //An array of fields to return.
        //If empty the default return fields will be from the active
listviewdefs.
        'select_fields' => array(
            'id',
            'name',
            'account_type',
            'phone_office',
            'assigned_user_name',
        ),

        //If the search is to only search modules participating in the
unified search.
        //Unified search is the SugarCRM Global Search alternative to
Full-Text Search.
        'unified_search_only' => false,

        //If only records marked as favorites should be returned.
        'favorites' => false
    );

    $search_by_module_result = $client->call('search_by_module',
$search_by_module_parameters);

    echo '<pre>';
    print_r($search_by_module_result);
    echo '</pre>';

?>
```

Result

```
stdClass Object
(
    [result_count] => 2
    [total_count] => 200
    [next_offset] => 2
```

```
    [entry_list] => Array
        (
            [0] => stdClass Object
                (
                    [id] => 18124607-69d1-b158-47ff-4f7cb69344f7
                    [module_name] => Leads
                    [name_value_list] => stdClass Object
                        (
                            [id] => stdClass Object
                                (
                                    [name] => id
                                    [value] =>
18124607-69d1-b158-47ff-4f7cb69344f7
                                )

                            [name] => stdClass Object
                                (
                                    [name] => name
                                    [value] => Bernie Worthey
                                )

                            [title] => stdClass Object
                                (
                                    [name] => title
                                    [value] => Senior Product Manager
                                )
                        )
                )

            [1] => stdClass Object
                (
                    [id] => 1cdfddc1-2759-b007-8713-4f7cb64c2e9c
                    [module_name] => Leads
                    [name_value_list] => stdClass Object
                        (
                            [id] => stdClass Object
                                (
                                    [name] => id
                                    [value] =>
1cdfddc1-2759-b007-8713-4f7cb64c2e9c
                                )

                            [name] => stdClass Object
                                (
                                    [name] => name
                                    [value] => Bobbie Kohlmeier
```

```
                                )

                            [title] => stdClass Object
                                (
                                    [name] => title
                                    [value] => Director Operations
                                )
                        )
                )
        )

    [relationship_list] => Array
        (
        )
)
```

Last Modified: 09/26/2015 04:23pm

Module Framework

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 03:22am

Introduction

Overview

The Sugar module framework.

About

A Sugar Module consists of the following files:

- A Vardefs file that specifies the Sugar metadata for the database table, fields, data types, and relationships.

- A SugarBean file that implements the functionality to create, retrieve, update, and delete objects in Sugar. SugarBean is the base class for all business objects in Sugar. Each module implements this base class with additional properties and methods specific to that module.
- Metadata files that define the contents and layout of the Sugar screens.
    - ListView: lists existing records in the module.
    - Detail View: displays record details.
    - EditView: allows user to edit the record.
    - SubPanels: displays the module's relationship with other Sugar modules.
    - Popups: displays list of records to link with another record.

Last Modified: 09/26/2015 04:23pm

MVC

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 03:22am

Introduction

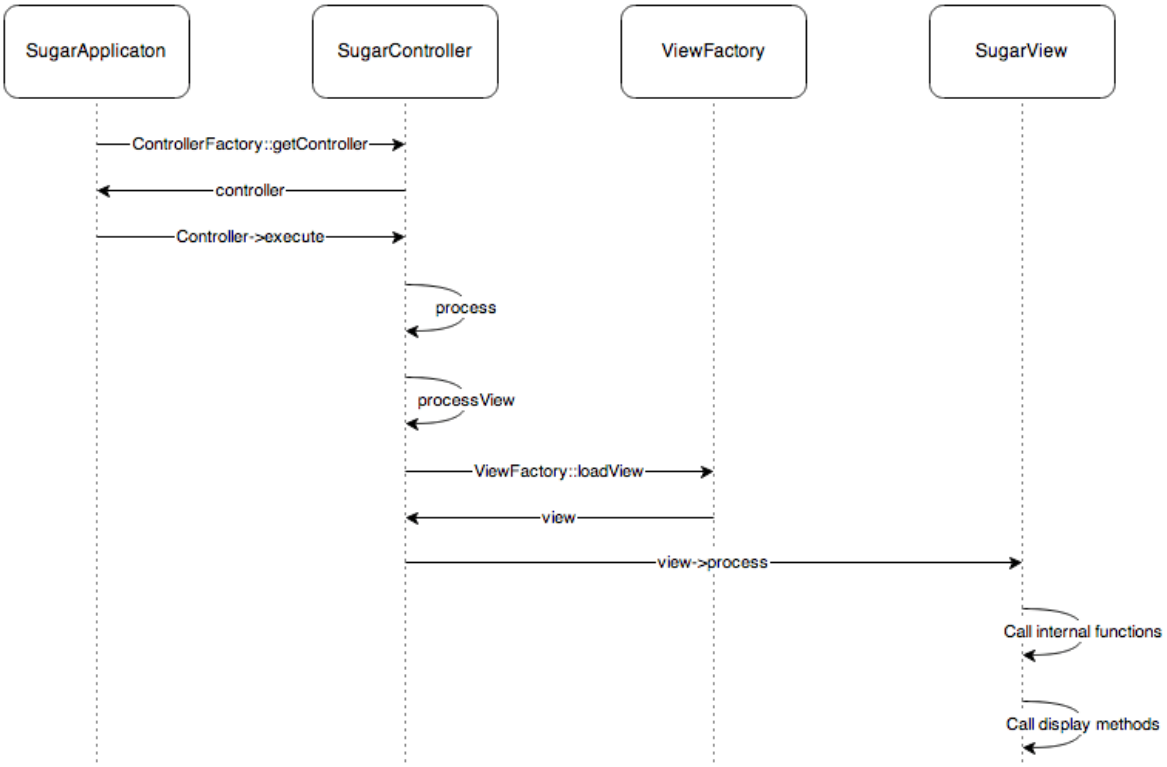
Model-View-Controller (MVC) Overview

A model-view-controller, or MVC, is a design philosophy that creates a distinct separation between business-logic and display logic.

- Model : This is the data object built by the business/application logic needed to present in the user interface. For Sugar, it is represented by the SugarBean and all subclasses of the SugarBean.
- View : This is the display layer which is responsible for rendering data from the Model to the end-user.
- Controller : This is the layer that handles user events such as "Save" and determines what business logic actions to take to build the model, and which view to load for rendering the data to end users.

SugarCRM MVC Implementation

The following is a sequence diagram that highlights some of the main components involved within the Sugar MVC framework.



Last Modified: 09/26/2015 04:23pm

Model

Overview

The Sugar Model is represented by the SugarBean, and any subclass of the SugarBean. Many of the common Sugar modules also use the SugarObjects class.

Sugar Object Templates

Sugar Objects extend the concept of subclassing a step further and allows you to subclass the vardefs. This includes inheriting of fields, relationships, indexes, and language files, but unlike subclassing, you are not limited to a single inheritance. If

there were a Sugar Object for fields used across every module such as id, deleted, or date_modified, you could have your module inherit from both Basic Sugar Object and the Person Sugar Object.

For example, the Basic type has a field 'name' with length 10 and Company has a field 'name' with length 20. If you inherit from Basic first then Company your field will be of length 20. Assuming you have defined the field 'name' in your module with length 60, then the module will always override any values provided by Sugar Objects.

There are six types of Sugar Object Templates:

- Basic : Contains the basic fields required by all Sugar modules
- Person : Based on the Contacts, Prospects, and Leads modules
- Issue : Based on the Bugs and Cases modules
- Company : Based on the Accounts module
- File : Based on the Documents module
- Sale : Based on the Opportunities module

We can take this a step further and add assignable to the mix. An assignable module would be one that can be assigned to users. Although this is not used by every module, many modules do let you assign records to users. SugarObject interfaces allow us to add "assignable" to modules in which we want to enable users to assign records.

SugarObject interfaces and SugarObject templates are very similar to one another, but the main distinction is that templates have a base class you can subclass while interfaces do not. If you look into the file structure you will notice that templates include many additional files including a full metadata directory. This is currently used primarily for Module Builder.

File Structure

- ./include/SugarObjects/interfaces
- ./include/SugarObjects/templates

Implementation

There are two things you need to do to take advantage of SugarObjects:

1) Your class needs to subclass the SugarObject class you wish to extend.

```
class MyClass extends Person{
    function MyClass()
    {
        parent::Person();
    }
}
```

2) In your vardefs.php file add the following to the end:

```
VardefManager::createVardef(
    'Contacts',
    'Contact',
    array(
        'default',
        'assignable',
        'team_security',
        'person'
    )
);
```

This tells the VardefManager to create a cache of the Contacts vardefs with the addition of all the default fields, assignable fields, team security fields (Sugar Professional and Enterprise only), and all fields from the person class.

Performance Considerations

VardefManager caches the generated vardefs into a single file that will be the file loaded at run time. Only if that file is not found will it load the vardefs.php file that is located in your modules directory. The same is true for language files. This caching also includes data for custom fields, and any vardef or language extensions that are dropped into the custom/ext framework.

Cache Files

- ./cache/modules/<module>/<object_name>vardefs.php
- ./cache/modules/<module>/languages/en_us.lang.php

Last Modified: 09/26/2015 04:23pm

View

Overview

Displays information to the browser.

Views / Actions

Views are not just limited to HTML data, you can have it send down JSON encoded data as part of the view or any other structure you wish. As with the controllers, there is a default class called SugarView which implements much of the basic logic for views, such as handling of headers and footers.
As a developer, to create a custom view you would place a view.<view_name>.php file in a views/ subdirectory within the module. For example, for the DetailView, you would create a file name view.detail.php and place this within the views/ subdirectory within the module. If a views subdirectory does not exist, you must create one.
In the file, create a class named: <Module>View<ViewName>. For example, for a list view within the Contacts module the class would be ContactsViewList. Note the first letter of each word is uppercase and all other letters are lowercase.
You can extend the class from SugarView, the parent class of all views, or you can extend from an existing view. For example, extending from the out-of-the-box list view can leverage a lot of the logic that has already been created for displaying a list view.

What are Views?

Views, otherwise known as actions are typically used to render views or to process logic. There are five main actions for a module:

- Display Actions
  - Detail View : A Detail View displays a read-only view of a particular record. Usually, this is accessed via the List View. The Detail View displays the details of the object itself and related items (sub-panels). Sub-panels are miniature List Views of items that are related to the parent object. For example, Tasks assigned to a Project, or Contacts for an Opportunity will appear in sub-panels in the Project or Opportunity Detail View.
    ./<module>/metadata/detailviewdefs.php defines a module's Detail View page layout. ./<module>/metadata/subpaneldefs.php defines the subpanels that are displayed in the module's Detail View page.
  - Edit View : The Edit View page is accessed when a user creates a new record or edits details of an existing one. Edit View can also be accessed directly from the List View. ./<module>/metadata/editviewdefs.php

defines a module's Edit View page layout.
    - List View : This Controller action enables the search form and search results for a module. Users can perform actions such as delete, export, update multiple records (mass update), and drill into a specific record to view and edit the details. Users can see this view by default when they click one of the module tabs at the top of the page. Files in each module describe the contents of the list and search view.
- Process Actions
    - Save : This Controller action is processed when the user clicks Save in the record's Edit View.
    - Delete : This action is processed when the user clicks Delete in the Detail View of a record or in the Detail View of a record listed in a sub-panel.

Methods

There are two main methods to over-ride within a view:

- preDisplay() : This performs pre-processing within a view. This method is relevant only for extending existing views. For example, the include/MVC/View/views/view.edit.php file uses it, and enables developers who wishes to extend this view to leverage all of the logic done in preDisplay() and either override the display() method completely or within your own display() method call parent::display().
- display() : This method displays the data to the screen. Place the logic to display output to the screen here.

Loading the View

The ViewFactory class tries to load the view for view in this sequence, and will use the first one it finds:

- ./custom/modules/<module>/views/view.<view>.php
- ./modules/<module>/views/view.<view>.php
- ./custom/include/MVC/View/view.<view>.php
- ./include/MVC/Views/view.<view>.php

Implementation

```
class ContactsViewList extends SugarView{
    function ContactsViewList()
    {
```

```
        parent::SugarView();
    }
    function display()
    {
        echo 'This is my Contacts ListView';
    }
}
```

File Structure

- ./include/MVC/Views/view.<view>.php
- ./custom/include/MVC/Views/view.<view>.php
- ./modules/<module>/views/view.<view>.php
- ./custom/modules/<module>/views/view.<view>.php
- ./include/MVC/Views/SugarView.php

Display Options for Views

The Sugar MVC provides developers with granular control over how the screen looks when a view is rendered. Each view can have a config file associated with it. So, in the example above, the developer would place a view.edit.config.php within the views/ subdirectory . When the EditView is rendered, this config file will be picked up. When loading the view, ViewFactory class will merge the view config files from the following possible locations with precedence order (high to low):

- ./customs/modules/<module>/views/view.<view>.config.php
- ./modules/<module>/views/view.<view>.config.php
- ./custom/include/MVC/View/views/view.<view>.config.php
- ./include/MVC/View/views/view.<view>.config.php

Implementation

The format of these files is as follows:

```
$view_config = array(
    'actions' =>
        array(
            'popup' => array(
                'show_header' => false,
                'show_subpanels' => false,
                'show_search' => false,
```

```
                'show_footer' => false,
                'show_JavaScript' => true,
            ),
        ),
    'req_params' => array(
            'to_pdf' => array(
                'param_value' => true,
                'config' => array(
                    'show_all' => false
                ),
            ),
        ),
);
```

To illustrate this process, let us take a look at how the 'popup' action is processed. In this case, the system will go to the actions entry within the view_config and determine the proper configuration. If the request contains the parameter to_pdf, and is set to be true then it will automatically cause the show_all configuration parameter to be set false, which means none of the options will be displayed.

Last Modified: 09/26/2015 04:23pm

Controller

Overview

Addresses the basic actions of a module.

Controllers

The main controller, named SugarController, addresses the basic actions of a module from EditView and DetailView to saving a record. Each module can override this SugarController by adding a controller.php file into its directory. This file extends the SugarController, and the naming convention for the class is: <module>Controller

Inside the controller you define an action method. The naming convention for the method is: action_<action name>

There are more fine grained control mechanisms that a developer can use to override the controller processing. For example if a developer wanted to create a new Save action there are three places where they could possibly override.

- action_save :This is the broadest specification and gives the user full control over the Save process.
- pre_save : A user could override the population of parameters from the form.
- post_save : This is where the view is being setup. At this point the developer could set a redirect url, do some post save processing, or set a different view.

Upgrade-Safe Implementation

You can also add a custom Controller that extends the module's Controller if such a Controller already exists. For example, if you want to extend the Controller for a module, you should check if that module already has a module-specific controller. If so, you extend from that controller class. Otherwise, you extend from SugarController class. In both cases, you should place the custom controller class file in ./custom/modules/<module>/Controller.php instead of the module directory. Doing so makes your customization upgrade-safe.

File Structure

- ./include/MVC/Controller/SugarController.php
- ./include/MVC/Controller/ControllerFactory.php
- ./modules/<MyModule>/Controller.php
- ./custom/modules/<MyModule>/controller.php

Implementation

If the module does not contain a controller.php file in ./modules/<module>/ you will create the following file:

./custom/modules/<module>/controller.php

```
class <module>Controller extends SugarController{
    function action_<action>()
    {
        $this->view = '<action lowercase>';
    }
}
```

If the module does contain a controller.php file, you will need to extend it by doing the following:

./custom/modules/<module>/controller.php

```
require_once('modules/<module>/controller.php');
class Custom<module>Controller extends <module>Controller{
    function action_<action>()
    {
        $this->view = '<action lowercase>';
    }
}
```

Mapping Actions to Files

You can choose not to provide a custom action method as defined above, and instead specify your mappings of actions to files in $action_file_map. Take a look at ./include/MVC/Controller/action_file_map.php as an example:

```
$action_file_map['subpanelviewer'] =
'include/SubPanel/SubPanelViewer.php';
$action_file_map['save2'] = 'include/generic/Save2.php';
$action_file_map['deleterelationship'] =
'include/generic/DeleteRelationship.php';
$action_file_map['import'] = 'modules/Import/index.php';
```

Here the developer has the opportunity to map an action to a file. For example Sugar uses a generic sub-panel file for handling subpanel actions. You can see above that there is an entry mapping the action 'subpanelviewer' to ./include/SubPanel/SubPanelViewer.php.

The base SugarController class loads the action mappings in the following path sequence:

- ./include/MVC/Controller
- ./modules/<module>
- ./custom/modules/<module>
- ./custom/include/MVC/Controller

Each one loads and overrides the previous definition if in conflict. You can drop a new action_file_map in the later path sequence that extends or overrides the mappings defined in the previous one.

Upgrade-Safe Implementation

If you want to add custom action_file_map.php to an existing module that came with the SugarCRM release, you should place the file at

./custom/modules/<module>/action_file_map.php

File Structure

- ./include/MVC/Controller/action_file_map.php
- ./modules/<module>/action_file_map.php
- ./custom/modules/<module>/action_file_map.php

Implementation

```
$action_file_map['soapRetrieve'] = 'custom/SoapRetrieve/soap.php';
```

Classic Support (Not Recommended)

Classic support allows you to have files that represent actions within your module. Essentially, you can drop in a PHP file into your module and have that be handled as an action. This is not recommended, but is considered acceptable for backward compatibility. The better practice is to take advantage of the action_<action> structure.

File Structure

- ./modules/<module>/<action>.php

Controller Flow Overview

For example, if a request comes in for DetailView the controller will handle the request as follows:

1. Start in index.php and load the SugarApplication instance.
2. SugarApplication instantiates SugarControllerFactory.
3. SugarControllerFactory loads the appropriate Controller.
4. Check for ./custom/modules/<module>/Controller.php.
    1. If not found, check for ./modules/<module>/Controller.php.
    2. If not found, load SugarController.php.
5. Call on the appropriate action.
    1. Look for ./custom/modules/<module>/<action>.php. If found and ./custom/modules/<module>/views/view.<action>.php is not found, use this view.
    2. If not found check for modules/<module>/<action>.php. If found and

./modules/<module>/views/view.<action>.php is not found, then use the ./modules/<module>/<action>.php action.
3. If not found, check for the method action_<action> in the controller.
4. If not found, check for an action_file_mapping.
5. If not found, report error "Action is not defined".

Last Modified: 01/15/2016 10:05pm

Examples

Provides an overview of example MVC customizations

Last Modified: 11/19/2015 03:22am

Changing the ListView Default Sort Order

Overview

This article addresses the need to customize the advanced search layout options to change the default sort order from ascending to descending.

Customization Information

This customization involves creating several custom files that extend the stock files and are detailed in following sections. Please note that when creating or moving files you will need to rebuild the file map. More information on rebuilding the file map can be found in the SugarAutoLoader .

Extending the Search Form

First, we will need to extend the SearchForm class. To do this, we will create a CustomSearchForm class that extends the original SearchForm class located in ./include/SearchForm/SearchForm2.php. We will then override the _displayTabs method to check the $_REQUEST['sortOrder'] and default it to descending if it isn't set.

./custom/include/SearchForm/SearchForm2.php

```
<?php
```

```
require_once 'include/SearchForm/SearchForm2.php';

class CustomSearchForm extends SearchForm
{
    /**
     * displays the tabs (top of the search form)
     *
     * @param string $currentKey key in $this->tabs to show as the
current tab
     *
     * @return string html
     */

    function _displayTabs($currentKey)
    {
        //check and set the default sort order
        if (!isset($_REQUEST['sortOrder']))
        {
            $_REQUEST['sortOrder'] = 'DESC';
        }

        return parent::_displayTabs($currentKey);;
    }
}

?>
```

Extending the List View

Next, we will need to extend the ListView. We will create a ViewCustomList class that extends the original ListView located in ./include/MVC/View/views/view.list.php. In the ViewCustomList class, we will override the prepareSearchForm and getSearchForm2 methods to call the CustomSearchForm class.

./custom/include/MVC/View/views/view.customlist.php

```
<?php

require_once 'include/MVC/View/views/view.list.php';

class ViewCustomList extends ViewList
{
```

```
    function prepareSearchForm()
    {
        $this->searchForm = null;

        //search
        $view = 'basic_search';

        if(!empty($_REQUEST['search_form_view']) &&
$_REQUEST['search_form_view'] == 'advanced_search')
            $view = $_REQUEST['search_form_view'];

        $this->headers = true;

        if(!empty($_REQUEST['search_form_only']) &&
$_REQUEST['search_form_only'])
            $this->headers = false;

        elseif(!isset($_REQUEST['search_form']) ||
$_REQUEST['search_form'] != 'false')
        {
            if(isset($_REQUEST['searchFormTab']) &&
$_REQUEST['searchFormTab'] == 'advanced_search')
            {
                $view = 'advanced_search';
            }

            else
            {
                $view = 'basic_search';
            }
        }

        $this->view = $view;
        $this->use_old_search = true;

        if (SugarAutoLoader::existingCustom('modules/' . $this->module
. '/SearchForm.html') &&
            !SugarAutoLoader::existingCustom('modules/' .
$this->module . '/metadata/searchdefs.php'))         {
            require_once('include/SearchForm/SearchForm.php');
            $this->searchForm = new SearchForm($this->module,
$this->seed);
        }
        else
        {
            $this->use_old_search = false;
```

```
            //Updated to require the extended CustomSearchForm class
            require_once('custom/include/SearchForm/SearchForm2.php');

            $searchMetaData =
SearchForm::retrieveSearchDefs($this->module);

            $this->searchForm = $this->getSearchForm2($this->seed,
$this->module, $this->action);
            $this->searchForm->setup($searchMetaData['searchdefs'],
$searchMetaData['searchFields'], 'SearchFormGeneric.tpl', $view,
$this->listViewDefs);
            $this->searchForm->lv = $this->lv;
        }
    }

    /**
     * Returns the search form object
     *
     * @return SearchForm
     */

    protected function getSearchForm2($seed, $module, $action =
"index")
    {
        //Updated to use the extended CustomSearchForm class
        return new CustomSearchForm($seed, $module, $action);
    }
}

?>
```

Extending the Sugar Controller

Finally, we will create a CustomSugarController class that extends the original SugarController located in ./include/MVC/Controller/SugarController.php. We will then need to override the do_action and post_action methods to execute their parent methods as well as the action_listview method to assign the custom view to the view attribute.

./custom/include/MVC/Controller/SugarController.php

```
<?php

/**
```

```
* Custom SugarCRM controller
* @api
*/

class CustomSugarController extends SugarController
{
    /**
     * Perform the specified action.
     * This can be overridden in a sub-class
     */

    private function do_action()
    {
        return parent::do_action();
    }

    /**
     * Perform an action after to the specified action has occurred.
     * This can be overridden in a sub-class
     */

    private function post_action()
    {
        return parent::post_action();
    }

    /**
     * Perform the listview action
     */

    protected function action_listview()
    {
        parent::action_listview();

        //set the new custom view
        $this->view = 'customlist';
    }
}

?>
```

Last Modified: 09/26/2015 04:23pm

Metadata

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 04:22am

Introduction

Overview

Overview of the metadata framework.

Metadata Framework

Background

Metadata is defined as information about data. In SugarCRM, metadata refers to the framework of using files to abstract the presentation and business logic found in the system. The metadata framework is described in definition files that are processed using PHP. The processing usually includes the use of Smarty templates for rendering the presentation, and JavaScript libraries to handle some business logic that affects conditional displays, input validation, and so on.

Application Metadata

All application modules are defined in the modules.php file. It contains several variables that define which modules are active and usable in the application.

The file is located under the '<sugar root>/include' folder. It contains the $moduleList() array variable which contains the reference to the array key to look up the string to be used to display the module in the tabs at the top of the application, the coding standard is for the value to be in the plural of the module name; for example, Contacts, Accounts, Widgets, and so on.

The $beanList() array stores a list of all active beans (modules) in the application. The $beanList entries are stored in a 'name' => 'value fashion with the 'name' value being in the plural and the 'value' being in the singular of the module name. The 'value' of a $beanList() entry is used to lookup values in our next modules.php variable, the $beanFiles() array.

The $beanFiles variable is also stored in a 'name' => 'value' fashion. The 'name', typically in singular, is a reference to the class name of the object, which is looked up from the $beanList 'value', and the 'value' is a reference to the class file.

The remaining relevant variables in the modules.php file are the $modInvisList variable which makes modules invisible in the regular user interface (i.e., no tab appears for these modules), and the $adminOnlyList which is an extra level of security for modules that are can be accessed only by administrators through the Admin page.

Module Metadata

The following table lists the metadata definition files found in the modules/[module]/metadata directory, and a brief description of their purpose within the system.

File	Description
additionalDetails.php	Used to render the popup information displayed when a user hovers the mouse cursor over a row in the List View.
editviewdefs.php	Used to render a record's EditView.
detailviewdefs.php	Used to render a record's DetailView.
listviewdefs.php	Used to render the List View display for a module.
metafiles.php	Used to override the location of the metadata definition file to be used. The EditView, DetailView, List View, and Popup code check for the presence of these files.
popupdefs.php	Used to render and handle the search form and list view in popups
searchdefs.php	Used to render a module's basic and advanced search form displays
sidecreateviewdefs.php	Used to render a module's quick create form shown in the side shortcut panel
subpaneldefs.php	Used to render a module's subpanels shown when viewing a record's DetailView

SearchForm Metadata

The search form layout for each module is defined in the module's metadata file searchdefs.php. A sample of the Account's searchdefs.php appears as:

```
<?php
$searchdefs['Accounts'] = array(
    'templateMeta' => array(
        'maxColumns' => '3',
        'widths' => array(
             'label' => '10',
             'field' => '30'
        )
    ),
    'layout' => array(
        'basic_search' => array(
            'name',
            'billing_address_city',
            'phone_office',
            array(
                'name' => 'address_street',
                'label' => 'LBL_BILLING_ADDRESS',
                'type' => 'name',
                'group' => 'billing_address_street'
            ),
            array(
                'name' => 'current_user_only',
                'label' => 'LBL_CURRENT_USER_FILTER',
                'type'=>'bool'
            ),
        ),
        'advanced_search' => array(
            'name',
            array(
                'name' => 'address_street',
                'label' => 'LBL_ANY_ADDRESS',
                'type' => 'name'
            ),
            array(
                'name' => 'phone',
                'label' => 'LBL_ANY_PHONE',
                'type' => 'name'
            ),
            'website',
            array(
                'name' => 'address_city',
```

```
                    'label' => 'LBL_CITY',
                    'type' => 'name'
                ),
                array(
                    'name' => 'email',
                    'label' =>'LBL_ANY_EMAIL',
                    'type' => 'name'
                ),
                'annual_revenue',
                array(
                    'name' => 'address_state',
                    'label' =>'LBL_STATE',
                    'type' => 'name'
                ),
                'employees',
                array(
                    'name' => 'address_postalcode',
                    'label' =>'LBL_POSTAL_CODE',
                    'type' => 'name'
                ),
                array(
                    'name' => 'billing_address_country',
                    'label' =>'LBL_COUNTRY',
                    'type' => 'name'
                ),
                'ticker_symbol',
                'sic_code',
                'rating',
                'ownership',
                array(
                    'name' => 'assigned_user_id',
                    'type' => 'enum',
                    'label' => 'LBL_ASSIGNED_TO',
                    'function' => array(
                        'name' =>'get_user_array',
                        'params' => array(false)
                    )
                ),
                'account_type',
                'industry',
            ),
        ),
    );
?>
```

The contents of the searchdefs.php file contains the Array variable $searchDefs with one entry. The key is the name of the module as defined in $moduleList array defined in include/modules.php. The value of the $searchDefs array is another array that describes the search form layout and fields.

The 'templateMeta' key points to another array that controls the maximum number of columns in each row of the search form ('maxColumns'), as well as layout spacing attributes as defined by 'widths'. In the above example, the generated search form files will allocate 10% of the width spacing to the labels and 30% for each field respectively.

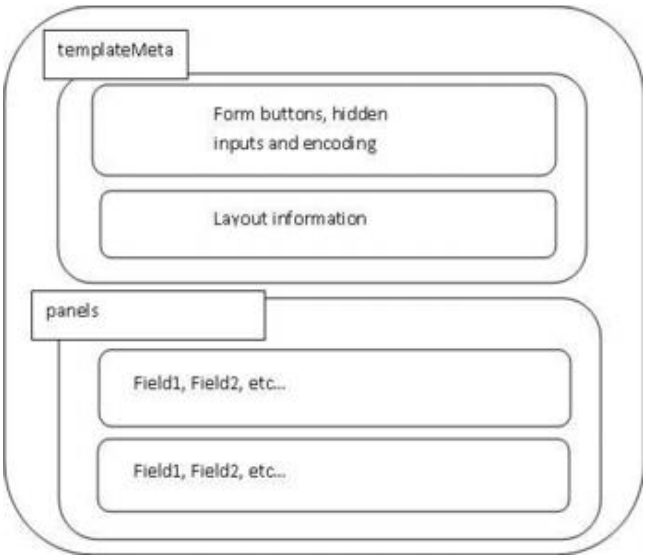
The 'layout' key points to another nested array which defines the fields to display in the basic and advanced search form tabs. Each individual field definition maps to a SugarField widget. See the SugarField widget section for an explanation about SugarField widgets and how they are rendered for the search form, DetailView, and EditView.

The searchdefs.php file is invoked from the MVC framework whenever a module's list view is rendered (see include/MVC/View/views/view.list.php). Within view.list.php checks are made to see if the module has defined a SearchForm.html file. If this file exists, the MVC will run in classic mode and use the aforementioned include/SearchForm/SearchForm.php file to process the search form. Otherwise, the new search form processing is invoked using include/SearchForm/SearchForm2.php and the searchdefs.php file is scanned for first under the custom/modules/[module]/metadata directory and then in modules/[module]/metadata.

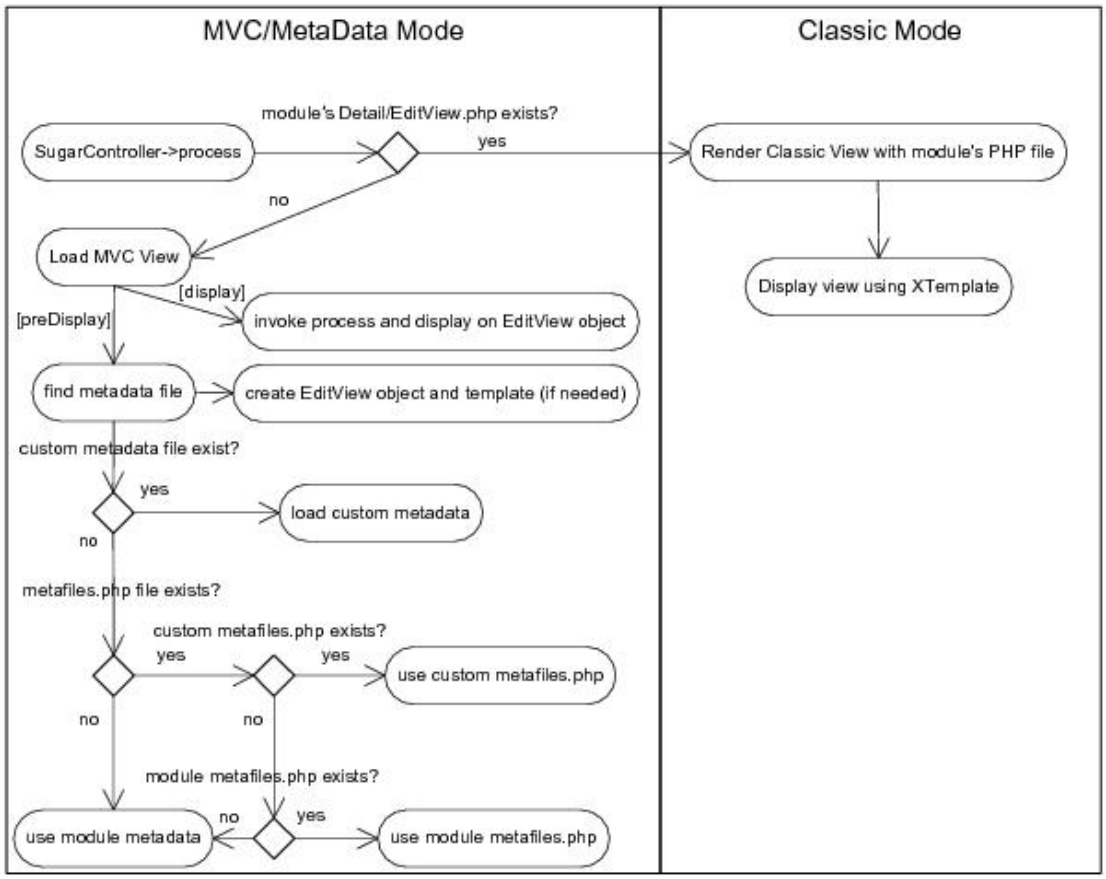
The processing flow for the search form using the metadata subpaneldefs.php file is similar to that of EditView and DetailView.

DetailView and EditView Metadata

Metadata files are PHP files that declare nested Array values that contain information about the view (buttons, hidden values, field layouts, etc.). A visual diagram that represents how the Array values declared in the Metadata file are nested is as follows:



The following diagram highlights the process of how the application determines which Metadata file is to be used when rendering a request for a view.



The "Classic Mode" on the right hand side of the diagram represents the SugarCRM pre-5.x rendering of a Detail/Editview. This section will focus on the MVC/Metadata mode.

When the view is first requested, the preDisplay method will attempt to find the correct Metadata file to use. Typically, the Metadata file will exist in the [root level]/modules/[module]/metadata directory, but in the event of edits to a layout through the Studio interface, a new Metadata file will be created and placed in the [root level]/custom/modules/[module]/metadata directory. This is done so that changes to layouts may be restored to their original state through Studio, and also to allow changes made to layouts to be upgrade-safe when new patches and upgrades are applied to the application. The metafiles.php file that may be loaded allows for the loading of Metadata files with alternate naming conventions or locations. An example of the metafiles.php contents can be found for the Accounts module (though it is not used by default in the application).

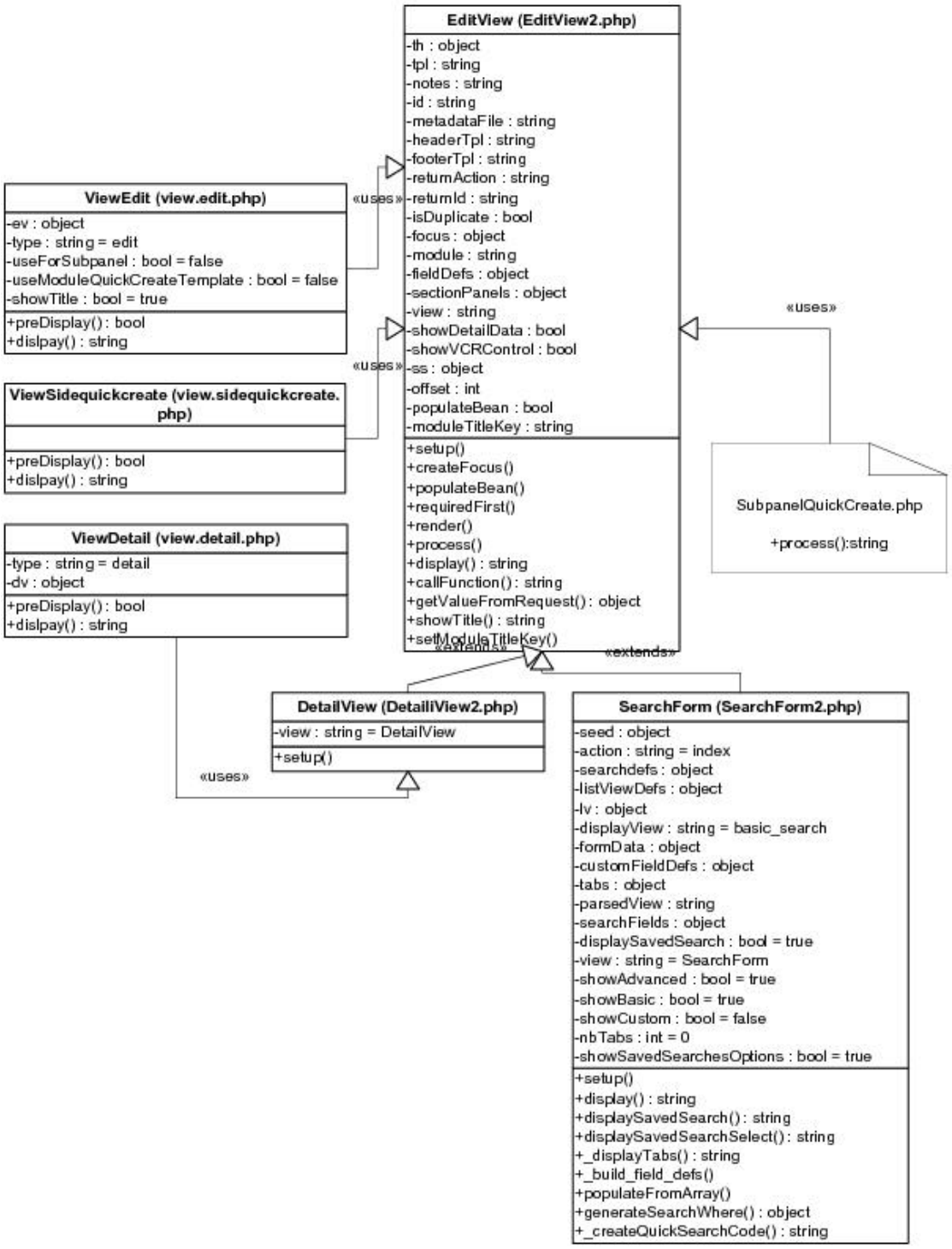
```
$metafiles['Accounts'] = array(
    'detailviewdefs' =>
'modules/Accounts/metadata/detailviewdefs.php',
    'editviewdefs' => 'modules/Accounts/metadata/editviewdefs.php',
    'ListViewdefs' => 'modules/Accounts/metadata/ListViewdefs.php',
    'searchdefs' => 'modules/Accounts/metadata/searchdefs.php',
    'popupdefs' => 'modules/Accounts/metadata/popupdefs.php',
    'searchfields' => 'modules/Accounts/metadata/SearchFields.php',
);
```

After the Metadata file is loaded, the preDisplay method also creates an EditView object and checks if a Smarty template file needs to be built for the given Metadata file. The EditView object does the bulk of the processing for a given Metadata file (creating the template, setting values, setting field level ACL controls if applicable, etc.). Please see the EditView process diagram for more detailed information about these steps.

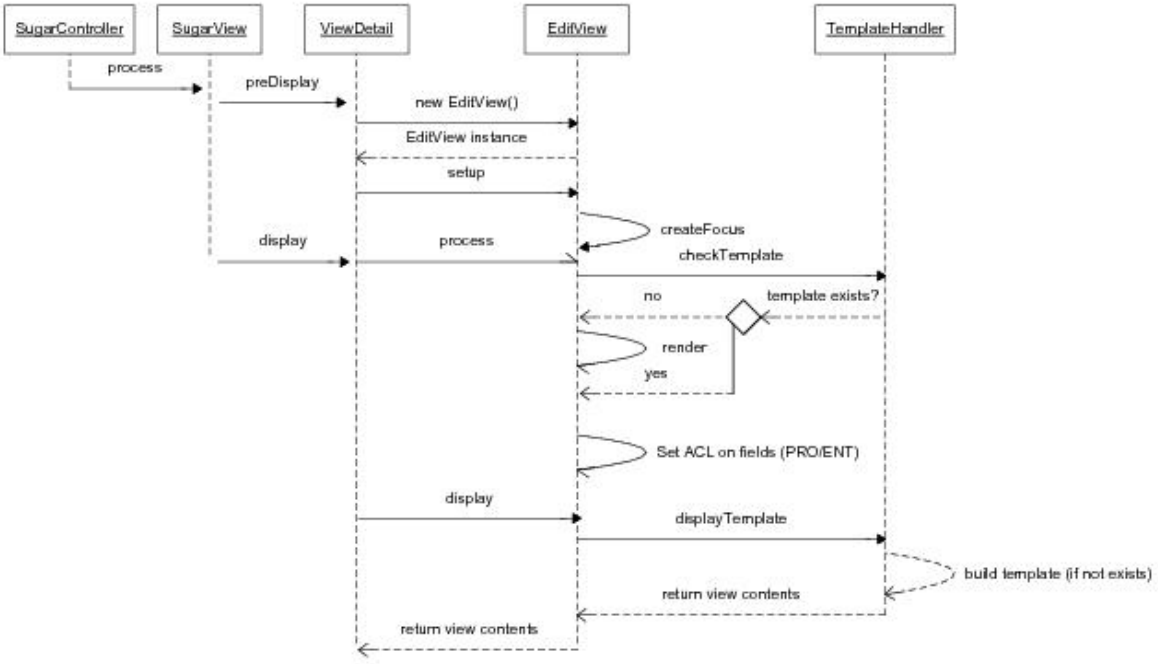
After the preDisplay method is called in the view code, the display method is called, resulting in a call to the EditView object's process method, as well as the EditView object's display method.

The EditView class is responsible for the bulk of the Metadata file processing and creating the resulting display. The EditView class also checks to see if the resulting Smarty template is already created. It also applies the field level ACL controls for the Sugar Ultimate, Enterprise, Corporate, and Professional editions.

The classes responsible for displaying the Detail View and SearchForm also extend and use the EditView class. The ViewEdit, ViewDetail and ViewSidequickcreate classes use the EditView class to process and display its contents. Even the file that renders the quick create form display (SubpanelQuickCreate.php) uses the EditView class. DetailView (in DetailView2.php) and SearchForm (in SearchForm2.php) extend the EditView class while SubpanelQuickCreate.php uses an instance of the EditView class. The following diagram highlights these relationships.



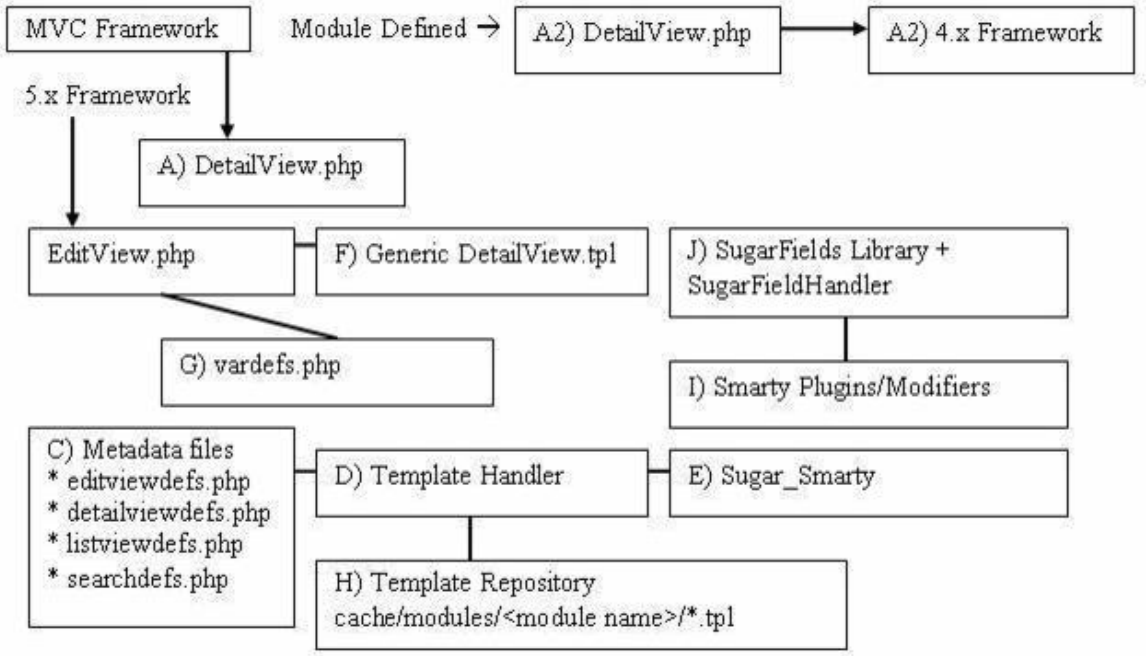
The following diagram highlights the EditView class's main responsibilities and their relationships with other classes in the system. We will use the example of a DetailView request although the sequence will be similar for other views that use the EditView class.



One thing to note is the EditView class's interaction with the TemplateHandler class. The TemplateHandler class is responsible for generating a Smarty template in the cache/modules/<module> directory. For example, for the Accounts module, the TemplateHandler will create the Smarty file cache/modules/Accounts/DetailView.tpl based on the Metadata file definition and other supplementary information from the EditView class. The TemplateHandler class actually uses Smarty itself to generate the resulting template that is placed in the aforementioned cache directory.

Some of the modules that are available in the SugarCRM application also extend the ViewDetail class. One example of this is the DetailView for the Projects module. As mentioned in the MVC section, it is possible to extend the view classes by placing a file in the modules/<module>/views directory. In this case, a view.detail.php file exists in the modules/Projects/views folder. This may serve as a useful example in studying how to extend a view and apply additional field/layout settings not provided by the EditView class.

The following diagram shows the files involved with the DetailView example in more detail.



A high level processing summary of the components for DetailViews follows:

The MVC framework receives a request to process the DetaiView.php (A) action for a module. For example, a record is selected from the list view shown on the browser with URL:

index.php?action=DetailView&module=Opportunities&record=46af9843-ccdf-f489-8833

At this point the new MVC framework checks to see if there is a DetailView.php (A2) file in the modules/Opportunity directory that will override the default DetailView.php implementation. The presence of a DetailView.php file will trigger the "classic" MVC view. If there is no DetailView.php (A2) file in the directory, the MVC will also check if you have defined a custom view to handle the DetailView rendering in MVC (that is,. checks if there is a file modules/Opportunity/views/view.detail.php). See the documentation for the MVC architecture for more information. Finally, if neither the DetailView.php (A2) nor the view.detail.php exists, then the MVC will invoke include/DetailView/DetailView.php (A).

The MVC framework (see views.detail.php in include/MVC/View/views folder) creates an instance of the generic DetailView (A)

```
// Call DetailView2 constructor
$dv = new DetailView2();
// Assign by reference the Sugar_Smarty object created from MVC
// We have to explicitly assign by reference to back support PHP 4.x
```

```
$dv->ss =& $this->ss;
// Call the setup function
$dv->setup($this->module, $this->bean, $metadataFile,
'include/DetailView/DetailView.tpl');
// Process this view
$dv->process();
// Return contents to the bufferecho
$dv->display();
```

When the setup method is invoked, a TemplateHandler instance (D) is created. A check is performed to determine which detailviewdefs.php metadata file to used in creating the resulting DetailView. The first check is performed to see if a metadata file was passed in as a parameter. The second check is performed against the custom/studio/modules/[Module] directory to see if a metadata file exists. For the final option, the DetailView constructor will use the module's default detailviewdefs.php metadata file located under the modules/[Module]/metadata directory. If there is no detailviewdefs.php file in the modules/[Module]/metadata directory, but a DetailView.html exists, then a "best guess" version is created using the metadata parser file in include/SugarFields/Parsers/DetailViewMetaParser.php (not shown in diagram).

The TemplateHandler also handles creating the quick search (Ajax code to do look ahead typing) as well as generating the JavaScript validation rules for the module. Both the quick search and JavaScript code should remain static based on the definitions of the current definition of the metadata file. When fields are added or removed from the file through the Studio application, this template and the resulting updated quick search and JavaScript code will be rebuilt.

It should be noted that the generic DetailView (A) defaults to using the generic DetailView.tpl smarty template file (F). This may also be overridden through the constructor parameters. The generic DetailView (A) constructor also retrieves the record according to the record id and populates the $focus bean variable.

The process() method is invoked on the generic DetailView.php instance:

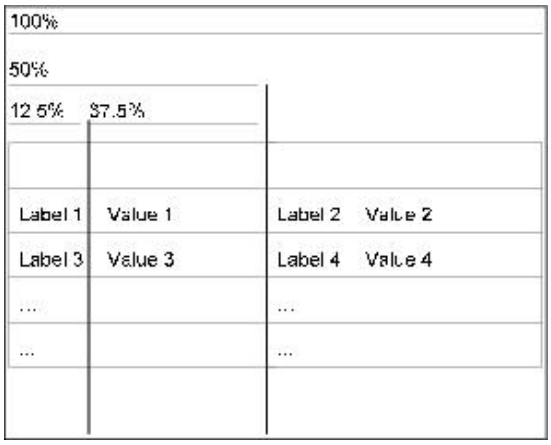
```
function process(){
    //Format fields first
    if($this->formatFields)
    {
        $this->focus->format_all_fields();
    }
        parent::process();
}
```

This in turn, calls the EditView->process() method since DetailView extends from EditView. The EditView->process() method will eventually call the

EditView->render() method to calculate the width spacing for the DetailView labels and values. The number of columns and the percentage of width to allocate to each column may be defined in the metadata file. The actual values are rounded as a total percentage of 100%. For example, given the templateMeta section's maxColumns and widths values:

```
'templateMeta' => array(
    'maxColumns' => '2',
    'widths' => array(
        array(
            'label' => '10',
             'field' => '30'
        ),
        array(
            'label' => '10',
             'field' => '30'
        )
    ),
),
```

We can see that the labels and fields are mapped as a 1-to-3 ratio. The sum of the widths only equals a total of 80 (10 + 30 x 2) so the actual resulting values written to the Smarty template will be at a percentage ratio of 12.5-to-37.5. The resulting fields defined in the metadata file will be rendered as a table with the column widths as defined:

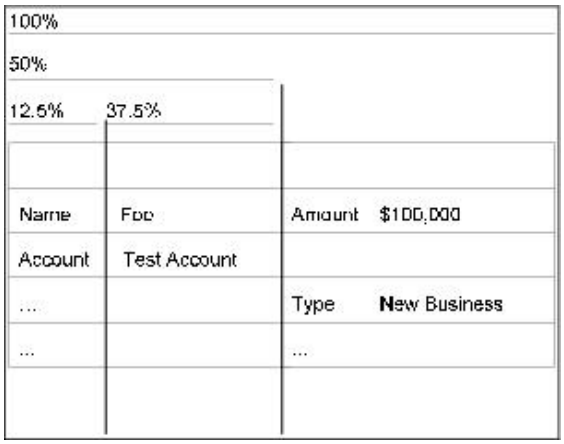


The actual metadata layout will allow for variable column lengths throughout the displayed table. For example, the metadata portion defined as:

```
'panels' => array(
    'default' => array(
        array(
            'name',
            array(
```

```
                'name' => 'amount',
                'label' => '{$MOD.LBL_AMOUNT} ({$CURRENCY})',
            ),
        ),
        array(
            'account_name',
        ),
        array(
            '',
            'opportunity_type',
        )
    )
)
```

This specifies a default panel under the panels section with three rows. The first row has two fields (name and amount). The amount field has some special formatting using the label override option. The second row contains the account_name field and the third row contains the opportunity_type column.



Secondly, the process() method populates the $fieldDefs array variable with the vardefs.php file (G) definition and the $focus bean's value. This is done by calling the toArray() method on the $focus bean instance and combining these value with the field definition specificed in the vardefs.php file (G).

The display() method is then invoked on the generic DetailView instance for the final step.
When the display() method is invoked, variables to the DetailView.tpl Smarty template are assigned and the module's HTML code is sent to the output buffer.

Before HTML code is sent back, the TemplateHandler (D) first performs a check to see if an existing DetailView template already exists in the cache respository (H). In this case, it will look for file cache/modules/Opportunity/DetailView.tpl. The operation of creating the Smarty template is expensive so this operation ensures

that the work will not have to be redone. As a side note, edits made to the DetailView or EditView through the Studio application will clear the cache file and force the template to be rewritten so that the new changes are reflected.

If the cache file does not exist, the TemplateHandler (D) will create the template file and store it in the cache directory. When the fetch() method is invoked on the Sugar_Smarty class (E) to create the template, the DetailView.tpl file is parsed.

Last Modified: 01/15/2016 10:21pm

SugarFields

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 03:22am

Introduction

Overview

Overview of SugarField widgets.

SugarField Widgets

SugarFields are the Objects that render the fields specified in the metadata (for example, your *viewdefs.php files). They can be found in include/SugarFields/Fields. In the directory include/SugarFields/Fields/Base you will see the files for the base templates for rendering a field for DetailView, EditView, ListView, and Search Forms. As well as the base class called SugarFieldBase.

File Structure

- ./include/SugarFields/Fields/
- ./include/SugarFields/Fields//DetailView.tpl
- ./modules/MyModule/vardefs.php

- ./modules/MyModule/metadata/defs.php

Implementation

This section describes the SugarFields widgets that are found in the ./include/SugarFields/Fields directory. Inside this folder you will find a set of directories that encapsulate the rendering of a field type (for example, Boolean, Text, Enum, and so on.). That is, there are user interface paradigms associated with a particular field type. For example, a Boolean field type as defined in a module's vardef.php file can be displayed with a checkbox indicating the boolean nature of the field value (on/off, yes/no, 0/1, etc.). Naturally there are some displays in which the rendered user interface components are very specific to the module's logic. In this example, it is likely that custom code was used in the metadata file definition. There are also SugarFields directories for grouped display values (e.g. Address, Datetime, Parent, and Relate).

Any custom code called by the metadata will be passed as unformatted data for numeric entries, and that custom code in the metadata will need individual handle formatting.

SugarFields widgets are rendered from the metadata framework whenever the MVC EditView, DetailView, or ListView actions are invoked for a particular module. Each of the SugarFields will be discussed briefly.

Most SugarFields will contain a set of Smarty files for abstract rendering the field contents and supporting HTML. Some SugarFields will also contain a subclass of SugarFieldBase to override particular methods so as to control additional processing and routing of the corresponding Smarty file to use. The subclass naming convention is defined as SugarField[Sugar Field Type] where the first letter of the Sugar Field Type should be in uppercase. The contents should also be placed in a corresponding .php file. For example, the contents of the enum type SugarField (rendered as in HTML) is defined in ./include/SugarFields/Fields/Enum/SugarFieldEnum.php. In that file, you can see how the enum type will use one of six Smarty template file depending on the view (edit, detail or search) and whether or not the enum vardef definition has a 'function' attribute defined to invoke a PHP function to render the contents of the field.

Last Modified: 11/19/2015 03:22am

Widgets

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 03:22am

Address

Overview

Renders various fields that together represent an address value.

Address

By default SugarCRM renders address values in the United States format:

```
Street
City, State Zip
```

The Smarty template layout defined in DetailView.tpl reflects this. Should you wish to customize the layout, depending on the $current_language global variable, you may need to add new files [$current_language].DetailView.tpl or [$current_language].EditView.tpl to the Address directory that reflect the language locale's address formatting.

Within the metadata definition, the Address field can be rendered with the snippet:

```
array (
    'name' => 'billing_address_street',
    'hideLabel' => true,
    'type' => 'address',
    'displayParams' => array(
        'key' => 'billing',
        'rows' => 2,
        'cols' => 30,
        'maxlength' => 150
    )
),
```

name	The vardefs.php entry to key field off of. Though not 100% ideal, we use the

	street value
hideLabel	Boolean attribute to hide the label that is rendered by metadata framework. We hide the billing_address_street label because the Address field already comes with labels for the other fields (city, state).
type	This is the field type override. billing_address_street is defined as a varchar in the vardefs.php file, but since we are interested in rendering an address field, we are overriding this here
displayParams	key - This is the prefix for the address fields. The address field assumes there are [key]_address_street, [key]_address_state, [key]_address_city, [key]_address_postalcode fields so this helps to distinguish in modules where there are two or more addresses (e.g. 'billing' and 'shipping'). rows, cols, maxlength - This overrides the default HTML <textarea> attributes for the street field component.

Note also the presence of file ./include/SugarFields/Fields/Address/SugarFieldAddress.js. This file is responsible for handling the logic of copying address values (from billing to shipping, primary to alternative, etc.). The JavaScript code makes assumptions using the key value of the grouped fields.
To customize various address formats for different locales, you may provide a locale specific implementation in the folder ./include/SugarFields/Fields/Address. There is a default English implementation provided. Locale implementations are system wide specific (you cannot render an address format for one user with an English locale and another format for another with a Japanese locale). Sugar locale settings are system-wide and the SugarField implementation reflects this. To modify based on a user's locale preferences is possible, but will require some customization.

Last Modified: 03/09/2016 03:43pm

Base

Overview

The default parent field.

Base

The Base field is the default parent field. It simply renders the value as is for detail views, and an HTML text field for edit views. All SugarFields that have a corresponding PHP file extend from SugarFieldBase.php or from one of the other SugarFields.

Last Modified: 11/19/2015 04:22am

Bool

Overview

Renders a checkbox to reflect the state of the value.

Bool

The Bool field is responsible for rendering a checkbox to reflect the state of the value. All boolean fields are stored as integer values. The Bool field will render a disabled checkbox for detail views. If the field value is "1" then the checkbox field will be checked. There is no special parameter to pass into this field from the metadata definition. As with any of the fields you have the option to override the label key string.

For example, in the metadata definition, the Boolean field do_not_call can be specified as:

```
'do_not_call'
```

or

```
array (
    array(
        'name' => 'do_not_call',
        'label' => 'LBL_DO_NOT_CALL' // Overrides label as defined in
```

```
vardefs.php
    )
),
```

Last Modified: 09/26/2015 04:23pm

Currency

Overview

Renders a checkbox to reflect the state of the value.

Currency

The Currency field is responsible for rendering a field that is formatted according to the user's preferences for currencies. This field's handles will format the field differently if the field name contains the text '_usd', this is used internally to map to the amount_usdollar fields which will always display the currency in the user's preferred currency (or the system currency if the user does not have one selected). If the field name does not contain '_usd', the formatting code will attempt to find a field in the same module called 'currency_id' and use that to figure out what currency symbol to display next to the formatted number. In order for this to work on list views and sub-panels, you will need to add the 'currency_id' column as a 'query_only' field, for further reference please see the Opportunities list view definitions.

Within the metadata definition, the Currency field can be rendered with the snippet:

```
// Assumes that amount is defined as a currency field in vardefs.php
file
'amount',
```

or

```
array(
    'name' => 'amount',
    'displayParams' => array('required' => true)
),
```

name	Standard name definition when metadata definition is defined as an array
displayParams	required (optional) - Marks the field as required and applies clients side validation to ensure value is present when save operation is invoked from edit view (Overrides the value set in vardefs.php). showFormats (optional) - Displays the user's date display preference as retrieved from the global $timedate variable's get_user_date_format() method.

Last Modified: 09/26/2015 04:23pm

Datetime

Overview

Renders an input text field along with an image to invoke the popup calendar picker.

Datetime

The Datetime field is responsible for rendering an input text field along with an image to invoke the popup calendar picker. This field differs from the Datetimecombo field in that there is no option to select the time values of a datetime database field.

Within the metadata definition, the Datetime field can be rendered with the snippet:

```
// Assumes that date_quote_expected_closed is defined as a datetime
field in vardefs.php file
'date_quote_expected_closed',
```

or

```
array(
    'name' => 'date_quote_expected_closed',
    'displayParams' => array(
        'required' => true,
        'showFormats' => true
    )
),
```

Last Modified: 09/26/2015 04:23pm

Datetimecombo

Overview

Renders a Datetime field with additional support to render dropdown lists for the hours and minutes.

Datetimecombo

The Datetimecombo field is similar to the Datetime field with additional support to render dropdown lists for the hours and minutes values, as well as a checkbox to enable/disable the entire field. The date portion (e.g. 12/25/2007) and time portion (e.g. 23:45) of the database fields are consolidated. Hence, the developer must take care to handle input from three HTML field values within the module class code. For example, in the vardefs.php definition for the date_start field in the Calls module:

```
'date_start' => array(
    'name' => 'date_start',
    'vname' => 'LBL_DATE',
    'type' => 'datetime',
    'required' => true,
    'comment' => 'Date in which call is schedule to (or did) start'
),
```

There is one database field, but when the Datetimecombo widget is rendered, it will produce three HTML fields for display- a text box for the date portion, and two dropdown lists for the hours and minutes values. The Datetimecombo widget will render the hours and menu dropdown portion in accordance to the user's $timedate preferences. An optional AM/PM meridiem drop down is also displayed should the user have selected a 12 hour base format (e.g. 11:00).

Within the metadata definition, the Datetimecombo field can be rendered with the snippet:

```
array(
    'name' => 'date_start',
    'type' => 'datetimecombo',
    'displayParams' => array(
        'required' => true,
        'updateCallback' => 'SugarWidgetScheduler.update_time();',
        'showFormats' => true,
        'showNoneCheckbox' => true
    ),
    'label'=>'LBL_DATE_TIME'
),
```

name	Standard name definition when metadata definition is defined as an array
type	metadata type override. By default, the field defaults to Datetime so we need to override it here in the definition.
displayParams	required (optional) - Marks the field as required and applies clients side validation to ensure value is present when save operation is invoked from edit view (Overrides the value set in vardefs.php). updateCallback (optional) - Defines custom JavaScript function to invoke when the values in the field are changed (date, hours or minutes). showFormats (optional) - Displays the user's date display preference as retrieved from the global $timedate variable's get_user_date_format() method. showNoneCheckbox (optional) - Displays a checkbox that when checked will disable all three field values
label (optional)	Standard metadata label override just to highlight this exhaustive example

Last Modified: 09/26/2015 04:23pm

Download

Overview

Renders the ability for a user to download a file.

Download

The File field renders a link that references the download.php file for the given displayParam['id'] value when in DetailView mode.

Within the metadata definition, the Download field can be rendered with the snippet:

```
array (
    'name' => 'filename',
    'displayParams' => array(
        'link' => 'filename',
        'id' => 'document_revision_id'
    )
)
```

name	Standard name definition when metadata definition is defined as an array
displayParams	required (optional) - Marks the field as required and applies clients side validation to ensure value is present when save operation is invoked from edit view (Overrides the value set in vardefs.php). id (required for detail view) - The field for which the id of the file will be opened via the download link. link (required for detail view) - The field for which the hyperlink value is

	displayed.

Last Modified: 02/01/2016 11:02pm

Enum

Overview

Renders an HTML <select> form element that allows for a single value to be chosen.

Enum

The size attribute of the <select> element is not defined so the element will render as a dropdown field in the EditView.

This field accepts the optional function override behavior that is defined at the vardefs.php file level. For example, in the Bugs module we have for the found_in_release field:

```
'found_in_release' => array(
    'name' => 'found_in_release',
    'type' => 'enum',
    'function' => 'getReleaseDropDown',
    'vname' => 'LBL_FOUND_IN_RELEASE',
    'reportable' => false,
    'merge_filter' => 'enabled',
    'comment' => 'The software or service release that manifested the
bug',
    'duplicate_merge' => 'disabled',
    'audited' => true,
),
```

The function override is not handled by the SugarFields library, but by the rendering code in ./include/EditView/EditView2.php.

Within the metadata definition, the Download field can be rendered with the snippet:

```
array (
    'name' => 'my_enum_field',
    'type' => 'enum',
```

```
    'displayParams' => array(
        'javascript' => 'onchange="alert(\'hello world!\')";'
    )
),
```

name	Standard name definition when metadata definition is defined as an array
displayParams	size (optional) – Controls the size of the field (affects SearchForm control on the browser only). Defaults to value of 6 for SearchForm. required (optional) - Marks the field as required and applies clients side validation to ensure value is present when save operation is invoked from edit view (Overrides the value set in vardefs.php). javascript (optional) - Custom JavaScript to embed within the <select> tag element (see above example for onchange event implementation).

Last Modified: 09/26/2015 04:23pm

File

Overview

Renders a file upload field.

File

The File field renders a file upload field in edit view, and a hyperlink to invoke download.php in detail view. Note that you will need to override the HTML form's enctype attribute to be "multipart/form-data" when using this field and handle the upload of the file contents in your code. This form enctype attribute should be set in the editviewdefs.php file. For example, for the Document's module we have the

form override:

```
$viewdefs['Documents']['EditView'] = array(
    'templateMeta' => array(
        'form' => array(
            'enctype' => 'multipart/form-data', // <--- override the
enctype
        ),
    )
)
```

Within the metadata file, the File field can be rendered with the snippet:

```
array (
    'name' => 'filename',
    'displayParams' => array(
        'link' => 'filename',
         'id' => 'document_revision_id'
    ),
),
```

name	Standard name definition when metadata definition is defined as an array
displayParams	required (optional) - Marks the field as required and applies clients side validation to ensure value is present when save operation is invoked from edit view (Overrides the value set in vardefs.php). id (required for detailviewdefs.php) - The record id that download.php will use to retrieve file contents link (required for detailviewdefs.php) - The text to display between the <a> </> tags. size (optional) - Affects edit view only. Override to set the display length attribute of the input field. maxlength(optional) - Affects edit view only. Override to set the display

	maxlength attribute of the input field (defaults to 255).

Last Modified: 09/26/2015 04:23pm

Float

Overview

Renders a field that is formatted as a floating point number.

Float

The Float field is responsible for rendering a field that is formatted as a floating point number. The precision specified will be followed, or the user's default precision will take over.

Within the metadata definition, the Float field can be rendered with the snippet:

```
// Assumes that weight is defined as a float field in vardefs.php file
'weight',
```

or

```
array(
    'name' => 'weight',
    'displayParams' => array(
        'precision' => 1
    )
),
```

name	Standard name definition when metadata definition is defined as an array
displayParams	required (optional) - Marks the field as required and applies clients side validation to ensure value is present when save operation is invoked from edit view (Overrides the value set in vardefs.php).

	precision (optional) – Sets the displayed precision of the field, this specifies the number of digits after the decimal place that the field will attempt to format and display. Some databases may further limit the precision beyond what can be specified here.

Last Modified: 09/26/2015 04:23pm

Html

Overview

Renders read-only content.

Html

The Html field is a simple field that renders read-only content after the content is run through the from_html method of ./include/utils/db_utils.php to encode entity references to their HTML characters (i.e. ">" = ">"). The rendering of the Html field type should be handled by the custom field logic within SugarCRM.

name	Standard name definition when metadata definition is defined as an array.
displayParams	none

Last Modified: 11/19/2015 03:22am

Iframe

Overview

Renders an iFrame field.

iFrame

In the detail view, the iFrame field creates an HTML element where the src attribute points to the given URL value supplied in the edit view. Alternatively, the URL may be generated from the values of other fields, and the base URL in the vardefs. If this is the case, the field is not editable in the edit view.

name	Standard name definition when metadata definition is defined as an array.
displayParams	None

Last Modified: 11/19/2015 04:22am

Image

Overview

Renders an <img> tag where the src attribute points to the value of the field.

Image

Similar to the Html field, the Image field simply renders a <img> tag where the src attribute points to the value of the field.

Within the metadata file, the Image field can be rendered with the snippet:

```
array (
    // The value of this is assumed to be some URL to an image file
    'name' => 'my_image_value',
    'type' => 'image',
    'displayParams' => array(
        'width' => 100,
        'length' => 100,
        'link' => 'http://www.cnn.com',
        'border' => 0
    ),
),
```

name	Standard name definition when metadata definition is defined as an array

displayParams	link (optional) - Hyperlink for the image when clicked on. width (optional) - Width of image for display. height (optional) - Height of image for display. border (optional) - Border thickness of image for display

Last Modified: 09/26/2015 04:23pm

Int

Overview

Renders a field that is formatted as a whole number.

Int

The Int field is responsible for rendering a field that is formatted as a whole number. This will always be displayed as a whole number, and any digits after the decimal point will be truncated.

Within the metadata definition, the Int field can be rendered with the snippet:

```
// Assumes that quantity is defined as a int field in vardefs.php file
'quantity',
```

or

```
array(
    'name' => 'quantity',
    'displayParams' => array(
        'required' => 1
    )
),
```

Last Modified: 09/26/2015 04:23pm

Link

Overview

Renders a hyperlink to a records detail view.

Link

The link field simply generates an <a> HTML tag with a hyperlink to the value of the field for detail views. For edit views, it provides the convenience of pre-filling the "http://" value. Alternatively, the URL may be generated from the values of other fields, and the base URL in the vardefs. If this is the case, the field is not editable in edit view.

Within the metadata file, the Link field can be rendered with the snippet:

```
array (
    // The value of this is assumed to be some URL to an image file
    'name' => 'my_image_value',
    'type' => 'link',
    'displayParams' => array(
        'title' => 'LBL_MY_TITLE'
    ),
),
```

name	Standard name definition when metadata definition is defined as an array
displayParams	required (optional) - Marks the field as required and applies clients side validation to ensure value is present when save operation is invoked from edit view (overrides the value set in vardefs.php). title (optional for detailviewdefs.php only) - The <a> tag's title attribute for browsers to display the link in their status area size (optional) - Affects edit view only.

	Override to set the display length attribute of the input field. maxlength(optional) - Affects edit view only. Override to set the display maxlength attribute of the input field (defaults to 255).

Last Modified: 09/26/2015 04:23pm

Multienum

Overview

Renders a bullet list of values for detail views.

Multienum

The Multienum fields renders a bullet list of values for detail views, and renders a <select> form element for edit views that allows multiple values to be chosen. Typically, the custom field handling in Sugar will map multienum types created through Studio, so you do not need to declare metadata code to specify the type override. Nevertheless, within the metadata file, the Multienum field can be rendered with the snippet:

```
array (
    'name' => 'my_multienum_field',
    'type' => 'multienum',
    'displayParams' => array(
        'javascript' => 'onchange="alert(\'hello world!\')";'
    )
),
```

name	Standard name definition when metadata definition is defined as an array
displayParams	size (optional) – Controls the size of the field (affects SearchForm control on browser only). Defaults to value of 6 for SearchForm.

	required (optional) - Marks the field as required and applies clients side validation to ensure value is present when save operation is invoked from edit view (Overrides the value set in vardefs.php). javascript (optional) - Custom JavaScript to embed within the <select> tag element (see above example for onchange event implementation).

Last Modified: 09/26/2015 04:23pm

Parent

Overview

Renders a dropdown for the parent module type.

Parent

The parent field combines a blend of a dropdown for the parent module type, and a text field with code to allow QuickSearch for quicksearch-enabled modules (see ./include/SugarFields/Fields/Parent/EditView.tpl file contents and JavaScript code for more information on enabling Quicksearch). There are also buttons to invoke popups and a button to clear the value. Because the parent field assumes proper relationships within the Sugar modules, it is not a field you can add through Studio or attempt to type override in the metadata files.

Last Modified: 11/19/2015 03:22am

Password

Overview

Renders an input text field for passwords.

Password

The password field simply renders an input text field with the type attribute set to "password" to hide user input values. It is available to edit views only.

name	Standard name definition when metadata definition is defined as an array
displayParams	required (optional) - Marks the field as required and applies clients side validation to ensure value is present when Save operation is invoked from edit view (Overrides the value set in vardefs.php). size (optional) - Override to set the display length attribute of the input field (defaults to 30).

Last Modified: 11/17/2015 11:37pm

Phone

Overview

Renders a callto:// URL to trigger VOIP applications.

Phone

The phone field simply invokes the callto:// URL references that could trigger Skype or other VOIP applications installed on the user's system. It is rendered for detail views only.

Last Modified: 11/19/2015 04:22am

Radioenum

Overview

Renders a group of radio buttons.

Radioenum

The Radioenum field renders a group of radio buttons. Radioenum fields are similar to the enum field, but only one value can be selected from the group.

Last Modified: 11/19/2015 06:52am

Readonly

Overview

Directs edit view calls to use the SugarField's detail view display.

Readonly

The readonly field simply directs edit view calls to use the SugarField's detail view display. There are no Smarty .tpl files associated with this field.

Last Modified: 11/19/2015 10:33pm

Relate

Overview

Combines a blend of a text field with code to allow quick search.

Relate

The Relate field combines a blend of a text field with code to allow quick search. The quicksearch code is generated for edit views (see ./include/TemplateHandler/TemplateHandler.php). There are also buttons to invoke popups and a button to clear the value. For detail views, the Relate field creates a hyperlink that will bring up a detail view request for the field's value.

Within the metadata file, the Relate field can be rendered with the snippet:

```
array (
    array(
        'name' => 'account_name',
        'type' => 'relate',
        'displayParams' => array(
            'required' => true
        )
    ),
),
```

This will create a relate field that allows the user to input a value not found in the quicksearch list.

name	Standard name definition when metadata definition is defined as an array
displayParams	required (optional) - Marks the field as required and applies clients side validation to ensure value is present when Save operation is invoked from edit view (Overrides the value set in vardefs.php). readOnly (optional for editviewdefs.php file only) - Makes the text field input area readonly so that you have to just use the popup selection. popupData - This field is generated for you by default. See include/SugarFields/Fields/SugarFieldRelate.php for more information. You should not need to override this setting. hideButtons (optional for editviewdefs.php SearchForm.php and popupdefs.php) – Hides the Select and Clear buttons normally displayed next to the editable Relate field.

Last Modified: 09/26/2015 04:23pm

Text

Overview

Renders a HTML form element for edit views.

Text

The Text field renders a HTML form element for edit views and displays the field value with newline characters converted to HTML elements in detail views.

Name	Standard name definition when metadata definition is defined as an array
displayParams	required (optional) - Marks the field as required and applies clients side validation to ensure value is present when Save operation is invoked from edit view (Overrides the value set in vardefs.php). maxlength (optional for editviewdefs.php file only) - Sets the maximum length of character input allowed in the field. rows (optional for editviewdefs.php file only) - Sets the number of rows in the field. cols (optional for editviewdefs.php file only) - Sets the number of cols in the field.

Last Modified: 11/19/2015 04:52am

Username

Overview

A helper field that assumes a salutation, first_name and last_name.

Username

The Username field is a helper field that assumes a salutation, first_name and last_name field exists for the vardefs of the module. It displays the three fields in the format:

```
{salutation} {first_name} {last_name}
```

Last Modified: 09/26/2015 04:23pm

Examples

Examples of working with metadata.

Last Modified: 09/26/2015 04:23pm

Adding QuickSearch to a custom field

Overview

How to add QuickSearch to a custom field.

Adding QuickSearch to a Custom Field

1. Include the default configs from QuickSearchDefaults.php. Most of the time you can use the predefined configurations and scripts.

   ```
   require_once('include/QuickSearchDefaults.php');
   $qsd = new QuickSearchDefaults();
   ```
2. Then set up the config for the input box you wish to have SQS. Account, Team, and User configs are available. The following configures SQS for account search on input box w/ id of 'parent_name', user and team in similar fashion with defaults.

   ```
   $sqs_objects = array(
        'parent_name' => $qsd->getQSParent(),
        'assigned_user_name' => $qsd->getQSUser(),
        'team_name' => $qsd->getQSTeam()
   ```

```
);
```

Notes on structure of config - replace the default parameters if they are different for the page.

- method : Unless you make a new method on the JSON server, keep this as query.
- populate_list : This defines the id's of the fields to be populated after a selection is made.

  QuickSearch will map the first item of field_list to the first item of populate_list. ie. field_list[0] = populate_list[0], field_list[1] = populate_list[1].... until the end of populate list.

- limit : reduce from 30 if query is large hit, but never less than 12.

- conditions : options are like_custom, contains, or default of starts with

  if using 'like_custom' also define 'begin'/'end' for strings to prepend or append to the user input

  disable: set this to true to disable SQS (optional, useful for disabling SQS on parent types in calls for example)

- post_onblur_function : this is an optional function to be called after the user has made a selection. It will be passed in an array with the items in field_list as keys and their corresponding values for the selection as values.

```
$sqs_objects = array (
    'account_name' => $qsd->getQSParent(),
    // the method on to use on the JSON server
    'method' => 'query',
    'modules' => array ('Accounts'), // modules to use
    'field_list' => array ('name', 'id'), // columns to select
    // id's of the html tags to populate with the columns.
    'populate_list' => array ('account_name', 'account_id'),
    // where clause, this code is for any account names that
have A WORD beginning with ...
    'conditions' =>
        array (array
('name'=>'name','op'=>'like_custom','end'=>'%','value'=>''),
        array ('name' => 'name', 'op' => 'like_custom',
'begin' => '% ', 'end' => '%', 'value' => '')),
    'group' => 'or', // grouping of the where conditions
    'order' => 'name', // ordering
    'limit' => '30', // number of records to pull
```

```
            'no_match_text' => $app_strings['ERR_SQS_NO_MATCH'], //
        text for no matching results
        ),
```

3. Include the necessary javascript files if sugar_grp1.js is not already loaded on the page.

```
    $quicksearch_js = '<script type="text/javascript" src="' .
getJSPath('include/javascript/sugar_grp1.js') . '"></script>';
```

4. Assign your config array to sqs_objects (important!)

```
    require_once('include/JSON.php');
$json = new JSON();$quicksearch_js .= '<script type="text/javascript"
language="javascript">sqs_objects = ' . $json->encode($sqs_objects) .
'</script>';
```

5. Add validation so that if there is no matching id for what the user has entered in an SQS field, an alert shows. This is for fields such as assigned users where the form must submit a valid ID.

```
    $javascript = new javascript();
$javascript->setFormName('EditView');$javascript->setSugarBean($this->
bean);
$javascript->addToValidateBinaryDependency('account_name', 'alpha',
app_strings['ERR_SQS_NO_MATCH_FIELD'] . $mod_strings['LBL_MEMBER_OF'],
'false', '', 'parent_id');
echo $javascript->getScript();
```

6. Add id tags and class name to the input box. Note that the input box must have class="sqsEnabled"!

```
    <input class="sqsEnabled"
       id="account_name"
       name='account_name'
       size='30'
       type='text'
       value="{ACCOUNT_NAME}">
<input id='account_id'
       name='account_id'
       type="hidden"
       value='{ACCOUNT_ID}'>
```

Having trouble? Take a look at the file ./module/Contacts/BusinessCard.php.

Last Modified: 09/26/2015 04:23pm

Creating a Custom Sugar Field

Overview

Creates a new SugarField for rendering a YouTube video.

Creating a Custom Sugar Field

In this example, we will use a custom text field in the Contacts module and then override the detail view of the custom field in the metadata file to link to our YouTube video.

The process is as follows:

1. Create a new TextField under Accounts named 'youtube_c'. You can do this by navigating to Admin > Studio > Accounts > Fields > Add Field.
2. Add this custom text field to both the edit view and detail view layouts. Save and deploy the updated layouts.
3. Create the directory ./custom/include/SugarFields/Fields/YouTube/. The name

of the directory (YouTube) corresponds to the name of the field type you are creating.

4. In the ./include/SugarFields/Fields/YouTube/ directory, create the file DetailView.tpl. For the detail view we will use the "embed" tag to display the video. In order to do this, you need to add the following to the template file:

./include/SugarFields/Fields/YouTube/DetailView.tpl

```
{if !empty({{sugarvar key='value' string=true}})}
    <object width="425" height="350">
        <param name="movie" value="http://www.youtube.com/v/{{sugarvar
key='value'}}></param>
        <param name="wmode" value="transparent"></param>
        <embed src="http://www.youtube.com/v/{{sugarvar key='value'}}"
type="application/x-shockwave-flash" wmode="transparent" width="425"
height="350"></embed>
    </object>{/if}
```

You will notice that we use the "{{" and "}}" double brackets around our variables. This implies that that section should be evaluated when we are creating the cache file. Remember that Smarty is used to generate the cached templates so we need the double brackets to distinguish between the stage for generating the template file and the stage for processing the runtime view.
Also note that will use the default edit view implementation that is provided by the base sugar field. This will give us a text field where people can input the YouTube video ID, so you do not need to create EditView.tpl. Also, we do not need to provide a PHP file to handle the SugarField processing since the defaults will suffice.

5. Now go to ./custom/modules/Accounts/metadata/detailview.php and add a type override to your YouTube field and save. In this example, the custom field is named "youtube_c".

```
array (
    'name' => 'youtube_c',
    'type' => 'YouTube',
    'label' => 'LBL_YOUTUBE',
),
```

6. Navigate to Admin > Repair > Quick Repair and Rebuild.
7. Your custom field is now ready to be displayed. You can now find the ID value of a YouTube video to insert into the edit view, and render the video in the detail view. An example you can try is "KDK1TmjmZsw".

* Remember to set your system to Developer Mode, or run a Quick Repair and

Rebuild to rebuild the cache directory.

Last Modified: 09/26/2015 04:23pm

Hiding the Quotes Module PDF Buttons

Overview

The PDF buttons on quotes are rendered differently than the standard buttons on most layouts. Since these buttons can't be removed directly from the DetailView in the detailviewdefs, the best approach is using jQuery to hide the buttons.

Hiding the PDF Buttons

This approach involves modifying the detailviewdefs.php in the custom/modules/Quotes/metadata directory to include a custom JavaScript file. If a custom detailviewdefs.php file doesn't exist, you will need to create it through Studio or by manually coping the detailviewdefs.php from the Quotes stock module metadata directory.

First, we will create a javascript file, say removePdfBtns.js, in the ./custom/modules/Quotes directory. This javascript file will contain the jQuery statements to hide the Quotes "Download PDF" and "Email PDF" buttons on the DetailView of the Quote.

./custom/modules/Quotes/removePdfBtns.js.

```
SUGAR.util.doWhen("typeof $ != 'undefined'", function(){
    YAHOO.util.Event.onDOMReady(function(){
        $("#pdfview_button").hide();
        $("#pdfemail_button").hide();
    });
});
```

Next, we will modify the custom detailviewdefs.php file to contain the 'includes' array element in the templateMeta array as follows:

./custom/modules/Quotes/metadata/detailviewdefs.php

```
$viewdefs['Quotes']['DetailView'] = array(

    'templateMeta' => array(
```

```
        'form' =>array(
        'closeFormBeforeCustomButtons' => true,

        'buttons'=> array(
            'EDIT',
            'DUPLICATE',
            'DELETE',
        ),

        'footerTpl' => 'modules/Quotes/tpls/DetailViewFooter.tpl'),
        'maxColumns' => '2',

        'widths' => array(
            array(
                'label' => '10',
                'field' => '30'
            ),

            array(
                'label' => '10',
                'field' => '30'
            )
        ),

        'includes' => array(
            array('file' => 'custom/modules/Quotes/removePdfBtns.js')
        ),
    ),
```

Finally, navigate to:

```
Admin > Repair > Quick Repair and Rebuild
```

The buttons will then be removed from the DetailView layouts.

Last Modified: 09/26/2015 04:23pm

Manipulating Buttons on Layouts

Overview

How to add custom buttons to the EditView and DetailView layouts.

Metadata

Before adding buttons to your layouts, you will need to understand how the metadata framework is used. Detailed information on the metadata framework can be found in the Metadata section.

Custom Layouts

Before you can add a button to your layout, you will first need to make sure you have a custom layout present. The stock layouts are located in ./modules/<module>/metadata/ and must be recreated in ./custom/modules/<module>/metadata/.

There are two ways to recreate a layout in the custom directory if it does not already exist. The first is to navigate to:

```
Studio > {Module} > Layouts > {View}
```

Once there, you can click the "Save & Deploy" button. This will create the layoutdef for you. Alternatively, you can also manually copy the layoutdef from the stock folder to the custom folder.

Editing Layouts

When editing layouts you have three options in having your changes reflected in the UI.

Developer Mode

You can turn on Developer Mode:

```
Admin > System Settings
```

Developer Mode will remove the caching of the metadata framework. This will cause your changes to be reflected when the page is refreshed. Make sure this setting is deactivated when you are finished with your customization.

Quick Repair and Rebuild

You can run a Quick Repair and Rebuild:

```
Admin > Repair > Quick Repair and Rebuild
```

Doing this will rebuild the cache for the metadata.

Saving & Deploying the Layout in Studio

You may also choose to load the layout in studio and then save & deploy it:

```
Admin > Studio > {Module} > Layouts > {View}
```

This process can be a bit confusing, however, once a layout is changed, you can then choose to load the layout in studio and then click "Save & Deploy" . This will rebuild the cache for that specific layout. Please note that any time you change the layout, you will have to reload the Studio layout view before deploying in order for this to work correctly.

Adding Custom Buttons

When adding buttons, there are several things to consider when determining how the button should be rendered. The following sections will outline these scenarios when working with the accounts editviewdefs located in ./custom/modules/Accounts/metadata/editviewdefs.php.

JavaScript Actions

If you are adding a button soley to execute JavaScript (no form submissions), you can do so by adding the button HTML to:

```
$viewdefs['<Module>']['<View>']['templateMeta']['form']['buttons']
```

Example

```
<?php

$viewdefs['Accounts'] =

array (
  'DetailView' =>

  array (
    'templateMeta' =>
```

```
    array (
      'form' =>

      array (
        'buttons' =>

        array (
          0 => 'EDIT',
          1 => 'DUPLICATE',
          2 => 'DELETE',
          3 => 'FIND_DUPLICATES',
          4 => 'CONNECTOR',
          5 =>

          array (
            'customCode' => '<input id="JavaScriptButton"
title="JavaScript Button" class="button" type="button"
name="JavaScriptButton" value="JavaScript Button"
onclick="alert(\'Button JavaScript\')">',
          ),
        ),
      ),
```

Submitting the Stock View Form

If you intend to submit the stock layout form ('formDetailView' or 'formEditView') to a new action, you can do so by adding a the button HTML with an onclick event as shown below to:

```
$viewdefs['<Module>']['<View>']['templateMeta']['form']['buttons']
```

Example

```
<?php

$viewdefs['Accounts'] =

array (
  'DetailView' =>

  array (
    'templateMeta' =>

    array (
```

```
      'form' =>

      array (
        'hidden' =>

        array (
          0 => '<input type="hidden" id="customFormField"
name="customFormField" value="">',
        ),

        'buttons' =>

        array (
          0 => 'EDIT',
          1 => 'DUPLICATE',
          2 => 'DELETE',
          3 => 'FIND_DUPLICATES',
          4 => 'CONNECTOR',
          5 =>

          array (
            'customCode' => '<input id="SubmitStockFormButton"
title="Submit Stock Form Button" class="button" type="button"
name="SubmitStockFormButton" value="Submit Stock Form Button"
onclick="var _form = document.getElementById(\'formDetailView\');
_form.customFormField.value = \'CustomValue\'; _form.action.value =
\'CustomAction\'; SUGAR.ajaxUI.submitForm(_form);">',
          ),
        ),
      ),
```

You should note in this example that there is also a 'hidden' index. This is where you should add any custom hidden inputs:

```
$viewdefs['<Module>']['<View>']['templateMeta']['form']['hidden']
```

Submitting Custom Forms

If you intend to submit a custom form, you will first need to set 'closeFormBeforeCustomButtons' to true. This will close out the current views form and allow you to create your own.

```
$viewdefs['<Module>']['<View>']['templateMeta']['form']['closeFormBefo
reCustomButtons']
```

Next, you will add the form and button HTML as shown below to:

```
$viewdefs['<Module>']['<View>']['templateMeta']['form']['buttons']
```

Example

```
<?php

$viewdefs['Accounts'] =

array (
  'DetailView' =>

  array (
    'templateMeta' =>

    array (
      'form' =>

      array (
        'closeFormBeforeCustomButtons' => true,
        'buttons' =>

        array (
          0 => 'EDIT',
          1 => 'DUPLICATE',
          2 => 'DELETE',
          3 => 'FIND_DUPLICATES',
          4 => 'CONNECTOR',
          5 =>

          array (
            'customCode' => '<form action="index.php" method="POST"
name="CustomForm" id="form"><input type="hidden"
name="customFormField" name="customFormField"
value="CustomValue"><input id="SubmitCustomFormButton" title="Submit
Custom Form Button" class="button" type="submit"
name="SubmitCustomFormButton" value="Submit Custom Form
Button"></form>',
          ),
        ),
      ),
```

Removing Buttons

To remove a button from the detail view will require modifying the ./modules/<module>/metadata/detailviewdefs.php.

The code is originally defined as:

```
$viewdefs[$module_name] = array (
    'DetailView' => array (
        'templateMeta' => array (
            'form' => array (
                'buttons' => array (
                    'EDIT',
                    'DUPLICATE',
                    'DELETE',
                    'FIND_DUPLICATES'
                ),
            ),
```

To remove one or more buttons, simply remove the 'buttons' attribute(s) that you do not want on the view.

```
$viewdefs[$module_name] = array (
    'DetailView' => array (
        'templateMeta' => array (
            'form' => array (
                'buttons' => array (
                    'DELETE',
                ),
            ),
```

Last Modified: 02/23/2016 05:25am

Manipulating Layouts Programmatically

Overview

How to manipulate and merge layouts programmatically.

The ParserFactory

The ParserFactory can be used to manipulate layouts such as editviewdefs or

detailviewdefs. This is a handy when creating custom plugins and needing to merge changes into an existing layout. The following example will demonstrate how to add a button to the detail view:

```
<?php

//Instantiate the parser factory for the Accounts DetailView.
require_once('modules/ModuleBuilder/parsers/ParserFactory.php');
$parser = ParserFactory::getParser('detailview', 'Accounts');

//Button to add
$button = array(
    'customCode'=>'<input type="button" name="customButton"
value="Custom Button">'
);

//Add button into the parsed layout
array_push($parser->_viewdefs['templateMeta']['form']['buttons'],
$button);

//Save the layout
$parser->handleSave(false);
```

Last Modified: 01/08/2017 04:38pm

Modifying Layouts to Display Additional Columns

Overview

By default, the editview, detailview, and quickcreate layouts for each module display two columns of fields. The number of columns to display can be customized on a per-module basis with the following steps.

Resolution

On-Demand

First, you will want to ensure your layouts are deployed in the custom directory. If you have not previously customized your layouts via Studio, go to Admin > Studio > {Module Name} > Layouts. From there, select each layout you wish to add additional columns to and click 'Save & Deploy'. This action will create a

corresponding layout file under the ./custom/modules/{Module Name}/metadata/ directory. The files will be named editviewdefs.php, detailviewdefs.php, and quickcreatedefs.php depending on the layouts deployed.

To access your custom files, go to Admin > Diagnostic Tool, uncheck all the boxes except for "SugarCRM Custom directory" and then click "Execute Diagnostic". This will generate an archive of your instance's custom directory to download, and you will find the layout files in the above path. Open the custom layout file, locate the 'maxColumns' value, and change it to the number of columns you would like to have on screen:

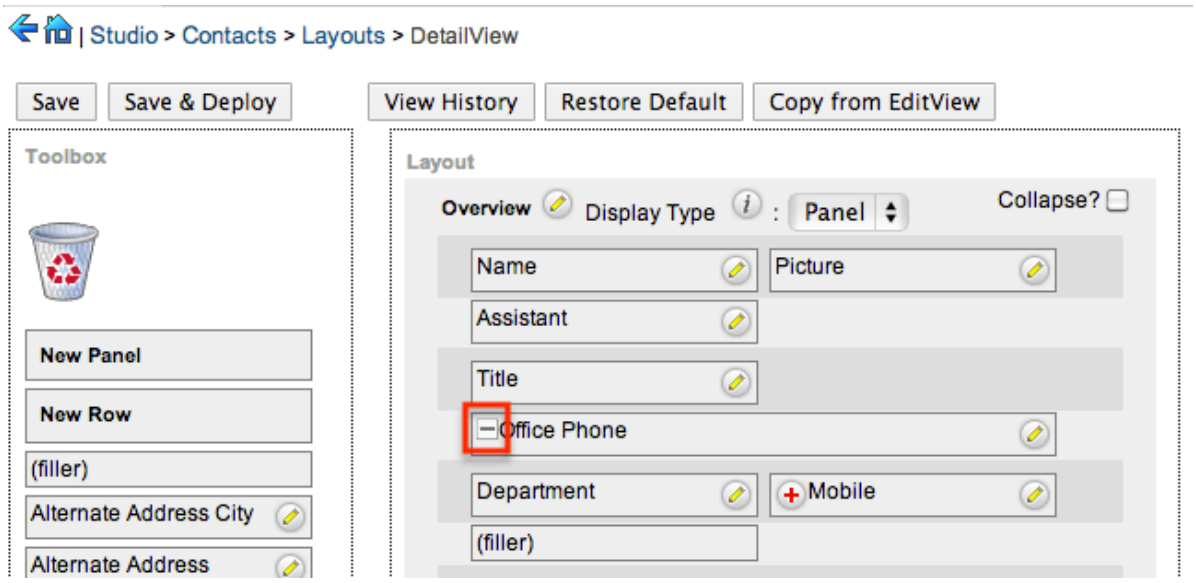
```
'maxColumns' => '3',
```

Once that is updated, locate the 'widths' array to define the spacing for your new column(s). You should have a label and field entry for each column in your layout:

```
'widths' => array (
    0 => array (
        'label' => '10',
        'field' => '30',
    ),

    1 => array (
        'label' => '10',
        'field' => '30',
    ),

    2 => array (
        'label' => '10',
        'field' => '30',
    ),
),
```

After this is completed, you will need to create a module-loadable package to install the changes on your On-Demand instance. More information on creating this package can be found in Creating an Installable Package That Copies Files . To upload and install the package, go to Admin > Module Loader.

Once the installation completes, you can navigate to Studio and add fields to your new column in the layout. For any rows that already contain two fields, the second field will automatically span the second and third column. Simply click the minus (-) icon to contract the field to one column and expose the new column space:



After you have added the desired fields in Studio, click 'Save & Deploy', and you are ready to go!

On-Site

First, you will want to ensure your layouts are deployed in the custom directory. If you have not previously customized your layouts via Studio, go to Admin > Studio > {Module Name} > Layouts. From there, select each layout you wish to add additional columns to and click 'Save & Deploy'. This action will create a corresponding layout file under the ./custom/modules/{Module Name}/metadata/ directory. The files will be named editviewdefs.php, detailviewdefs.php, and quickcreatedefs.php depending on the layouts deployed.

Next, open the custom layout file, locate the 'maxColumns' value, and change it to the number of columns you would like to have on screen:

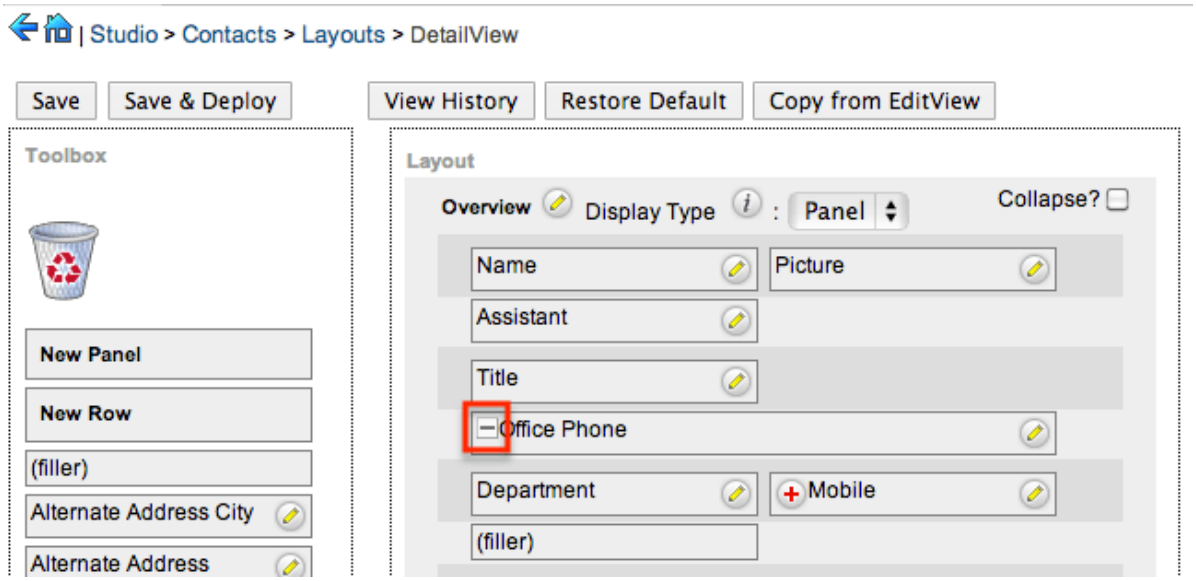
```
'maxColumns' => '3',
```

Once that is updated, locate the 'widths' array to define the spacing for your new column(s). You should have a label and field entry for each column in your layout:

```
'widths' => array (
    0 => array (
        'label' => '10',
        'field' => '30',
    ),

    1 => array (
```

```
        'label' => '10',

        'field' => '30',
    ),

    2 => array (
        'label' => '10',
        'field' => '30',
    ),
),
```

Once this is completed, you can navigate to Studio and add fields to your new column in the layout. For any rows that already contain two fields, the second field will automatically span the second and third column. Simply click the minus (-) icon to contract the field to one column and expose the new column space:



After you have added the desired fields in Studio, click 'Save & Deploy', and you are ready to go!

Last Modified: 09/26/2015 04:23pm

Vardefs

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 04:52am

Introduction

Overview

Vardefs (Variable Definitions) are used to provide the Sugar application with information about SugarBeans.

Vardefs

Vardefs specify information on the individual fields, relationships between beans, and the indexes for a given bean. Each module that contains a SugarBean file has a vardefs.php file located in it, which specifies the fields for that SugarBean. For example, the Vardefs for the Contact bean is located in ./modules/Contacts/vardefs.php.

Vardef files create an array called "$dictionary", which contains several entries including tables, fields, indices, and relationships.

Dictionary Array

- table : The name of the database table (usually, the name of the module) for this bean that contains the fields
- audited : Defines whether the module has field level auditing enabled
- fields : A list of fields and their attributes (see below)
- indices : A list of indexes that should be created in the database (see below)
- optimistic_locking : Set to true if this module should obey optimistic locking. Optimistic locking uses the modified date to ensure that the bean you are working on has not been modified by anybody else when you try and save to prevent loss of data.
- unified_search : Set to true if this module is available to be searched thru the Global Search, defaults to false. Has no effect if none of the fields in the fields array have the 'unified_search' attribute set to true.
- unified_search_default_enabled : Set to true if this module should be searched by default for new users thru the Global Search, defaults to true. Has no effect if 'unified_search' is not set to true.

Fields Array

The fields array contains one array for each field in the SugarBean. At the top level

of this array the key is the name of the field, and the value is an array of attributes about that field.

The list of possible attributes are as follows:

- name : The name of the field
- vname : The language pack ID for the label of this field
- type : The type of the attribute
  - assigned_user_name : A linked user name
  - bool : A boolean value
  - char : A character array
  - date : A date value with no time
  - datetime : A date and time
  - email : An email address field
  - enum : An enumeration (dropdown list from the language pack)
  - id : A database GUID
  - image : A photo-type field
  - link : A link through an explicit relationship
  - name : A non-db field type that concatenates other field values
  - phone : A phone number field to utilize with callto:// links
  - relate : Related bean
  - team_list : A team-based ID
  - text : A text area field
  - url : A hyperlinked field on the detailview
  - varchar : A variable-sized string
- table : The database table the field comes from. This is only used when needing to joing fields from another table outside of the module in focus.
- isnull : Whether the field can contain a null value
- len : The length of the field (number of characters if a string)
- options : The name of the enumeration (dropdown list) in the language pack for the field
- dbtype : The database type of the field (if different than the type)
- reportable : Determines whether the field will be available in the Reports and Workflow modules (for commercial editions)
- default : The default value for this field
- massupdate : Set to false if you do not want this field to show up in the mass update panel at the bottom of a module list view. Some field types are restricted from mass update regardless of this setting.
- rname : (for relate-type fields only) The field from the related variable that has the text
- id_name : (for relate-type fields only) The field from the bean that stores the ID for the related bean
- source : Set to 'non-db' if the field value does not come from the database. This can be used for calculated values or values retrieved in some other way.
- sort_on : The field to sort by if multiple fields are used in the presentation of

field's information.
- fields : (for concatenated values only) An array containing the fields that are concatenated
- db_concat_fields : (for concatenated values only) An array containing the fields to concatenate in the database
- unified_search : Set to true if this field should be searched through Global Search. Has no effect if the dictionary array setting 'unified_search' is not set to true.
- enable_range_search : (for date, datetime, and numeric fields only) Set to true if this field should support range searches in the Basic and Advanced Search layouts of the module.
- dependency : Allows a field to have a predefined formula to control the field's visibility
- studio :Controls the visibility of the field in the Studio editor. If set to false, then the field will not appear in any studio screens for the module. Otherwise, you may specify an Array of view keys for which the field's visibility should be removed from (ex: array('listview'=>false) will hide the field in the listview layout screen).

The following example illustrates a standard ID field for a Bean.

```
'id' => array (
    'name' => 'id',
    'vname' => 'LBL_ID',
    'type' => 'id',
    'required' => true,
),
```

Indices Array

This array contains a list of arrays that are used to create indexes in the database. The fields in this array are:

- name : The name of the index. This must be unique in most databases.
- type : Thy type of index (primary, unique, index)
- fields : The fields to index. This is an ordered array.
- source : Set to 'non-db' if you are creating an index for added application functionality such as duplication checking on imports.

The following example is to create a primary index called 'userspk' on the 'id' column

```
array(
    'name' => 'userspk',
    'type' => 'primary',
    'fields'=> array('id')
```

```
),
```

Relationships Array

The relationships array is used to specify relationships between Beans. Like the Indices array entries, it is a list of names with array values.

- lhs_module : The module on the left hand side of the relationship
- lhs_table : The table on the left hand side of the relationship
- lhs_key : The primary key column of the left hand side of the relationship
- rhs_module : The module on the right hand side of the relationship
- rhs_table : The table on the right hand side of the relationship
- rhs_key : The primary key column of the right hand side of the relationship
- relationship_type : The type of relationship ('one-to-many' or 'many-to-many')
- relationship_role_column : The type of relationship role
- relationship_role_column_value : Defines the unique identifier for the relationship role

The following example creates a reporting relationship between a contact and the person that they report to. The reports_to_id field is used to map to the id field in the contact of the person they report to. This is a one-to-many relationship in that each person is only allowed to report to one person. Each person is allowed to have an unlimited number of direct reports.

```
'contact_direct_reports' => array(
    'lhs_module' => 'Contacts',
    'lhs_table' => 'contacts',
    'lhs_key' => 'id',
    'rhs_module' => 'Contacts',
    'rhs_table' => 'contacts',
    'rhs_key' => 'reports_to_id',
    'relationship_type' => 'one-to-many'
),
```

Extending Vardefs

More information about extending and overriding vardefs can be found in the extensions framework under Vardefs .

Last Modified: 01/25/2016 09:53am

Examples

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 03:22am

Manually Creating Custom Fields

Overview

An overview of alternate ways to create custom fields without using studio.

Using ModuleInstaller

There are two ways to create a field using the ModuleInstaller class. This can be used both for an installer package and programmatically. An example of creating a field from a module loadable package can be found under Package Examples in the Creating an Installable Package that Creates New Fields section. Alternatively, you can programmatically add your custom fields by using the ModuleInstaller class with the install_custom_fields() method.

An example is below:

```
<?php

    $fields = array (
        //Text
        array(
            'name' => 'text_field_example',
            'label' => 'LBL_TEXT_FIELD_EXAMPLE',
            'type' => 'varchar',
            'module' => 'Accounts',
            'help' => 'Text Field Help Text',
            'comment' => 'Text Field Comment Text',
            'default_value' => '',
            'max_size' => 255,
            'required' => false, // true or false
            'reportable' => true, // true or false
            'audited' => false, // true or false
            'importable' => 'true', // 'true', 'false', 'required'
```

```
            'duplicate_merge' => false, // true or false
        ),

        //DropDown
        array(
            'name' => 'dropdown_field_example',
            'label' => 'LBL_DROPDOWN_FIELD_EXAMPLE',
            'type' => 'enum',
            'module' => 'Accounts',
            'help' => 'Enum Field Help Text',
            'comment' => 'Enum Field Comment Text',
            'ext1' => 'account_type_dom', //maps to options - specify
list name
            'default_value' => 'Analyst', //key of entry in specified
list
            'mass_update' => false, // true or false
            'required' => false, // true or false
            'reportable' => true, // true or false
            'audited' => false, // true or false
            'importable' => 'true', // 'true', 'false' or 'required'
            'duplicate_merge' => false, // true or false
        ),

        //MultiSelect
        array(
            'name' => 'multiselect_field_example',
            'label' => 'LBL_MULTISELECT_FIELD_EXAMPLE',
            'type' => 'multienum',
            'module' => 'Accounts',
            'help' => 'Multi-Enum Field Help Text',
            'comment' => 'Multi-Enum Field Comment Text',
            'ext1' => 'account_type_dom', //maps to options - specify
list name
            'default_value' => 'Analyst', //key of entry in specified
list
            'mass_update' => false, // true or false
            'required' => false, // true or false
            'reportable' => true, // true or false
            'audited' => false, // true or false
            'importable' => 'true', // 'true', 'false' or 'required'
            'duplicate_merge' => false, // true or false
        ),

        //Checkbox
        array(
            'name' => 'checkbox_field_example',
```

```
        'label' => 'LBL_CHECKBOX_FIELD_EXAMPLE',
        'type' => 'bool',
        'module' => 'Accounts',
        'default_value' => true, // true or false
        'help' => 'Bool Field Help Text',
        'comment' => 'Bool Field Comment',
        'audited' => false, // true or false
        'mass_update' => false, // true or false
        'duplicate_merge' => false, // true or false
        'reportable' => true, // true or false
        'importable' => 'true', // 'true', 'false' or 'required'
    ),

    //Date
    array(
        'name' => 'date_field_example',
        'label' => 'LBL_DATE_FIELD_EXAMPLE',
        'type' => 'date',
        'module' => 'Accounts',
        'default_value' => '',
        'help' => 'Date Field Help Text',
        'comment' => 'Date Field Comment',
        'mass_update' => false, // true or false
        'required' => false, // true or false
        'reportable' => true, // true or false
        'audited' => false, // true or false
        'duplicate_merge' => false, // true or false
        'importable' => 'true', // 'true', 'false' or 'required'
    ),

    //DateTime
    array(
        'name' => 'datetime_field_example',
        'label' => 'LBL_DATETIME_FIELD_EXAMPLE',
        'type' => 'datetime',
        'module' => 'Accounts',
        'default_value' => '',
        'help' => 'DateTime Field Help Text',
        'comment' => 'DateTime Field Comment',
        'mass_update' => false, // true or false
        'enable_range_search' => false, // true or false
        'required' => false, // true or false
        'reportable' => true, // true or false
        'audited' => false, // true or false
        'duplicate_merge' => false, // true or false
        'importable' => 'true', // 'true', 'false' or 'required'
```

```
        ),

        //Encrypt
        array(
            'name' => 'encrypt_field_example',
            'label' => 'LBL_ENCRYPT_FIELD_EXAMPLE',
            'type' => 'encrypt',
            'module' => 'Accounts',
            'default_value' => '',
            'help' => 'Encrypt Field Help Text',
            'comment' => 'Encrypt Field Comment',
            'reportable' => true, // true or false
            'audited' => false, // true or false
            'duplicate_merge' => false, // true or false
            'importable' => 'true', // 'true', 'false' or 'required'
        ),
    );

    require_once('ModuleInstall/ModuleInstaller.php');
    $moduleInstaller = new ModuleInstaller();
    $moduleInstaller->install_custom_fields($fields);
```

You can add in the labels for your custom fields by creating a corresponding language extension file and running a Quick Repair and Rebuild:

./custom/Extension/modules/Accounts/Ext/Language/en_us.<name>.php

```
<?php

    $mod_strings['LBL_TEXT_FIELD_EXAMPLE'] = 'Text Field Example';
    $mod_strings['LBL_DROPDOWN_FIELD_EXAMPLE'] = 'DropDown Field
Example';
    $mod_strings['LBL_CHECKBOX_FIELD_EXAMPLE'] = 'Checkbox Field
Example';
    $mod_strings['LBL_MULTISELECT_FIELD_EXAMPLE'] = 'Multi-Select
Field Example';
    $mod_strings['LBL_DATE_FIELD_EXAMPLE'] = 'Date Field Example';
    $mod_strings['LBL_DATETIME_FIELD_EXAMPLE'] = 'DateTime Field
Example';
    $mod_strings['LBL_ENCRYPT_FIELD_EXAMPLE'] = 'Encrypt Field
Example';
```

Using the Vardef Extensions

You should try to avoid creating your own custom fields using the vardefs as there

are several caveats:

- If your installation does not already contain custom fields, you must manually create the custom table. Otherwise the system will not recognize your fields custom vardef. This situation is outlined in the section below.

- You will have to run a Quick Repair and Rebuild, then execute the generated SQL after the vardef is installed.
- You have to correctly define the properties of a vardef. If you miss any, your field may not work properly.
- Your field name should end with an "_c" and should have the property 'source' set as 'custom_fields'. This is required as you should not modify core tables in Sugar and it is not permitted on On-Demand.
- Your vardef should specify the exact indexes of the properties you want to set. The example being to use: $dictionary['<module singular>']['fields']['example_c']['name'] = 'myfield_c'; instead of $dictionary['<module singular>']['fields']['example_c'] = array(['name' => 'myfield_c');. This will help prevent the system from losing any properties when loading from the extension framework.

The initial problem with creating your own custom vardef is to get the system to recognize the vardef and generate the database field. The problem will be illustrated with the example below:

./custom/Extension/modules/<module>/Ext/Vardefs/<file>.php

```
<?php

$dictionary['<module singular>']['fields']['example_c']['name'] =
'example_c';
$dictionary['<module singular>']['fields']['example_c']['vname'] =
'LBL_EXAMPLE_C';
$dictionary['<module singular>']['fields']['example_c']['type'] =
'varchar';
$dictionary['<module singular>']['fields']['example_c']['enforced'] =
'';
$dictionary['<module singular>']['fields']['example_c']['dependency']
= '';
$dictionary['<module singular>']['fields']['example_c']['required'] =
false,
$dictionary['<module singular>']['fields']['example_c']['massupdate']
= '0';
$dictionary['<module singular>']['fields']['example_c']['default'] =
'';
```

```
$dictionary['<module singular>']['fields']['example_c']['no_default']
= false,
$dictionary['<module singular>']['fields']['example_c']['comments'] =
'Example Varchar Vardef';
$dictionary['<module singular>']['fields']['example_c']['help'] = '';
$dictionary['<module singular>']['fields']['example_c']['importable']
= 'true';
$dictionary['<module
singular>']['fields']['example_c']['duplicate_merge'] = 'disabled';
$dictionary['<module
singular>']['fields']['example_c']['duplicate_merge_dom_value'] = '0';
$dictionary['<module singular>']['fields']['example_c']['audited'] =
false,
$dictionary['<module singular>']['fields']['example_c']['reportable']
= true,
$dictionary['<module
singular>']['fields']['example_c']['unified_search'] = false,
$dictionary['<module
singular>']['fields']['example_c']['merge_filter'] = 'disabled';
$dictionary['<module singular>']['fields']['example_c']['calculated']
= false,
$dictionary['<module singular>']['fields']['example_c']['len'] =
'255';
$dictionary['<module singular>']['fields']['example_c']['size'] =
'20';
$dictionary['<module singular>']['fields']['example_c']['id'] =
'example_c';
$dictionary['<module
singular>']['fields']['example_c']['custom_module'] = '';
//required to create the field in the _cstm table
$dictionary['<module singular>']['fields']['example_c']['source'] =
'custom_fields';
```

Once we have the vardef in place, we will need to consider if the custom field is for a module that doesn't currently have any other custom fields. If there are not any existing custom fields, we will need to create a corresponding record in fields_meta_data that will trigger the comparison process.

```
INSERT INTO fields_meta_data (id, name, vname, comments,
custom_module, type, len, required, deleted, audited, massupdate,
duplicate_merge, reportable, importable) VALUES ('<module>example_c',
'example_c', 'LBL_EXAMPLE_C', 'Example Varchar Vardef', '<module>',
'varchar', 255, 0, 0, 0, 0, 0, 1, 'true');
```

Finally, we can now navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions. After the repair, you will notice a section

at the bottom stating that there are differences between the database and vardefs. You will need to execute the scripts generated to create your custom field:

Missing <module>_cstm Table:

```
/*Checking Custom Fields for module : <module>*/

CREATE TABLE <module>_cstm (id_c char(36)  NOT NULL  , PRIMARY KEY
(id_c)) CHARACTER SET utf8 COLLATE utf8_general_ci;
```

Missing Columns:

```
/*MISSING IN DATABASE - example_c -  ROW*/

ALTER TABLE <module>_cstm add COLUMN example_c varchar(255)  NULL ;
```

Last Modified: 01/15/2016 10:09pm

Specifying Custom Indexes for Import Duplicate Checking

Overview

When performing imports, there is a predefined set of fields that you can specify to verify duplicate entries against. This article covers how to add an additional field or set of fields to verify duplicates against on imports.

Resolution

The import duplicate checking operates based on indices defined for that module. You can create a non-database index to check for a field. It is important that it is non-database as single column indices on your database can hamper overall performance. The following is an example to add the home phone field to the Contacts duplicate checking. You would first create the following file from the root directory of your Sugar installation on your web server:

./custom/Extension/modules/Contacts/Ext/Vardefs/custom_import_index.php

The name of the file is unimportant as long as it ends in a .php extension. The rest of the directory path above is case sensitive so please be sure to create the directories as shown. If you are creating the import index for a module other than

Contacts, then please substitute the corresponding directory name with the desired module. Please ensure that the entire directory path and file have the correct ownership and sufficient permissions for the web server to access the file. The contents of the file would be akin to the following code:

```
<?php
    $dictionary['Contact']['indices'][] = array(
         'name' => 'idx_home_phone_cstm',
         'type' => 'index',
         'fields' => array(
             0 => 'phone_home',
         ),
         'source' => 'non-db',
    );
```

It is important to note that the module name entered in line 2 of the above code is singular in the code (i.e. Contact, not Contacts). If you are unsure of what to enter for the module name, you can verify the name by opening the ./cache/modules/<module_name>/<module_name>vardefs.php file. The second line of that file will have text like the following:

```
$GLOBALS["dictionary"]["Contact"] = array (
```

The parameter following 'dictionary' is the exact same parameter you should use in the file defining your custom index. If you want to verify duplicates against a combination of fields (i.e. duplicates will only be flagged if the values of multiple fields match those of an existing record), then you can just add the desired fields to the 'fields' array in the code above.

Once this file is created, log into Sugar as an administrator and go to:

```
Admin > Repair > Quick Repair & Rebuild
```

After that process completes, the custom index should be available to utilize for duplicate verification when importing records in your module.

Last Modified: 09/26/2015 04:23pm

Language

Overview

How to work with the module language framework. More information on working

with the application language framework can be found in the Language section under Application Framework.

Language Keys

As Sugar is fully internationalized and localizable, each language is defined by a unique language key. The language keys will prefix any files dealing with languages. An example of a language key is 'en_us' which is used for English (US). For more information regarding please refer to the Language Keys section.

Module Label Framework

$mod_strings

The module language strings are stored in $mod_strings. This section will outline how the $mod_strings are compiled. All modules, whether out-of-box or custom will have an initial set of language files in the following directory:

./modules/<module>/language/<language key>.lang.php

As you begin working within the system and modifying labels through Studio, any changes to these labels will be reflected in the corresponding modules custom directory:

./custom/modules/<module>/language/<language key>.lang.php

Customizing Labels

If you are developing a customization and want to be able to create new or override existing label values, you will need to work within the extension modules directory. To do this you will create a file as follows:

./custom/Extension/modules/<module>/Ext/Language/<language key>.<unique_name>.php

The file will contain your override values. Please note that within this file you will set each label index individually. An example of this is:

```
<?php

    $mod_strings['LBL_KEY'] = 'My Display Label';
```

Once the file is created with your adjustments, you will then navigate to Admin > Repair > Quick Rebuild & Repair. This will compile all of the Extension files from:

./custom/Extension/modules/<module>/Ext/Language/

To:

./custom/modules/<module>/Ext/Language/<language key>.lang.ext.php

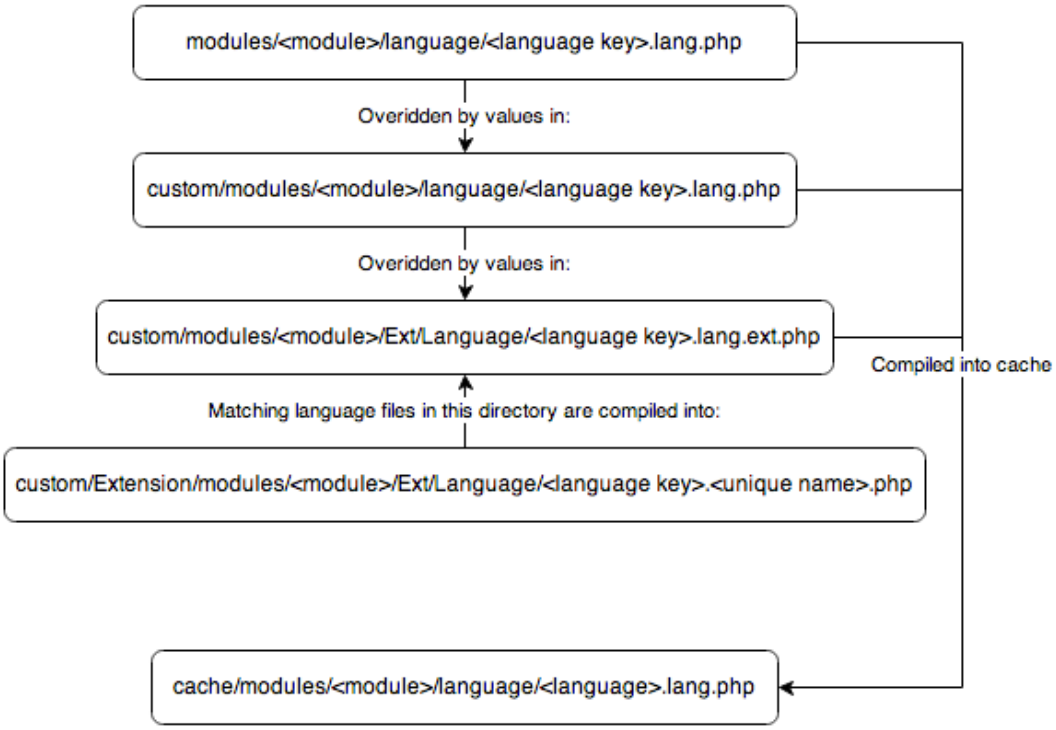
Label Cache

The file locations discussed above will all be compiled into the cache directory.

./cache/modules/<module>/language/<language key>.lang.php

The cached results of these files are what make up each modules '$mod_strings' definition.

Hierarchy Diagram



Retrieving Labels

There are two ways to retrieve a label. The first is to use the 'translate' function found in 'include/utils.php'. In using this function, the label will be retrieved for the current users language. This function can also be used to retrieve labels from mod_strings, app_strings or app_list_strings.

An example of this is:

```
require_once('include/utils.php');
$label = translate('LBL_KEY', 'Accounts');
```

Alternatively, you can also use the global variable $mod_strings as follows:

```
global $mod_strings;

$label = '';
if (isset($mod_strings['LBL_KEY']))
{
    $label = $mod_strings['LBL_KEY'];
}
```

Accessing Language Pack Strings with JavaScript

All language pack strings are accessible within the browser-side JavaScript. Use the following JavaScript call to access these strings:

```
// LBL_LIST_LAST_NAME string stored in Contacts $mod_strings
SUGAR.language.get('Contacts', 'LBL_LIST_LAST_NAME');
```

These JavaScript language files are cached. Rebuild the JavaScript files from the Repair console in the Admin section if the language files are changed. This removes the cache files and rebuilds the JavaScript files when needed. It also increments js_lang_version in sugar_config to recache the JavaScript files in the user's browser.

Last Modified: 09/26/2015 04:23pm

Relationships

Overview

Provides technical overview and know-how of a technology related to the product

or a particular feature.

Relationships

In the ./metadata directory, all the many-to-many relationships are defined and included in the $dictionary array. The files are stored in ./metadata/MetaData.php. Tables are generated based on these definitions. These files are included through the ./modules/TableDictionary.php. If you create a custom many-to-many relationship, you will need to add the reference to the new relationship by adding the new reference in the file custom/application/Ext/TableDictionary/tabledictionary.php. You may need to create this file if it does not exist. These changes will take effect after you clear the Sugar Cache by running the "Quick Repair and Rebuild" option from the Admin Repair screen.
The following are the definitions for $dictionary[]. They are similar to the Vardefs. If necessary use that page as a reference as well.

- : The index for this relationship in the $dictionary array
- table : The name of the table that is created in the database
- fields : Array containing arrays for each column definition. The join table must have a field for the primary key of each table to be linked, a primary key for the join table itself, a deleted field for relationship unlinking, and a date_modified field to track when the relationship changes. Additional fields can be added to track the role played by the relationship,
- indices : The database indices on the relationship table
- relationships : Definitions of the relationships between the two tables
    - lhs_modules : The left hand module. Should match $beanList index
    - lhs_table : The left hand table name
    - lhs_key : The key to use from the left table
    - rhs_modules : The right hand module. Should match $beanList index
    - rhs_table : The right hand table name
    - rhs_key : The key to use from the right table
    - relationship_type : Relationship type
    - join_table : Join table used to join items
    - join_key_lhs : Left table key. Should exist in table field definitions above
    - join_key_rhs : Right table key. Should exist in table field definitions above

Note that relationship metadata and the vardefs are the critical building blocks of the new list views. In conjunction with [module]/metadata/ListViewdefs.php, these three elements generate your list view.

Last Modified: 11/19/2015 03:22am

Subpanels

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 04:52am

Introduction

Overview

How subpanels work.

Subpanels

Subpanels are used in a given module's DetailView to display relationships with other modules. Examples include the Contacts subpanel under the Opportunities module's DetailView, or the Tasks subpanel for the Projects module.

The references below are the areas in the code where subpanels are generated from. Running rebuild relationships in the repair section in the admin panel is necessary after making changes to or creating subpanels.

The module that contains the subpanel is where the vardefs array index is defined. There is an index referring to the module that will appear as the subpanel of type 'link'.

One-to-Many Relationships

For Accounts, the reference necessary for the Cases subpanel is defined as follows in the ./modules/Accounts/vardefs.php

```
'cases' => array (
    'name' => 'cases',
    'type' => 'link',
    'relationship' => 'account_cases', //relationship definition is
below
```

```
    'module' => 'Cases',
    'bean_name' => 'aCase',
    'source' => 'non-db',
    'vname' => 'LBL_CASES',
),
```

For a one-to-many, the 'relationship' index defined above must also be in the vardefs.

```
'account_cases' => array(
    'lhs_module' => 'Accounts',
    'lhs_table' => 'accounts',
    'lhs_key' => 'id',
    'rhs_module' => 'Cases',
    'rhs_table' => 'cases',
    'rhs_key' => 'account_id',
    'relationship_type' => 'one-to-many'
),
```

Since this is a one-to-many, there is no need for a relationship table, which is only defined in the ./metadata directory.

Many-to-Many Relationships

For Accounts, the reference necessary for the Bugs subpanel is defined as follows in the ./modules/Accounts/vardefs.php

```
'bugs' => array (
    'name' => 'bugs',
    'type' => 'link',
    'relationship' => 'accounts_bugs', //relationship table
    'module' => 'Bugs',
    'bean_name' => 'Bug',
    'source' => 'non-db',
    'vname' => 'LBL_BUGS',
),
```

Since this is many-to-many relationship, and there already exists a relationship table, there is no need to define the relationship in the vardefs. However, the relationship metadata must be defined as shown below.

Relationship Metadata

If you have a many-to-many relationship, a table must exist for the relationship.

For a new relationship, you must add the details of the relationship file (accounts_bugsMetaData.php in this example) to TableDictionary.php in the /modules directory. You must then run Repair Database to create the new relationships table (accounts_bugs in the example below). To remain upgrade-safe you must put your custom changes into ./custom/application/ext/tabledictionary/tabledictionary.ext.php.
In the ./metadata directory, the relationship must exist and included in the $dictionary array. To keep consistent with the above accounts_bugs example, here is the content of the accounts_bugsMetaData.php

```
$dictionary['accounts_bugs'] = array(
    //the table that is created in the database
    'table' => 'accounts_bugs',
    'fields' => array(
        // id for relationship
        array(
            'name' => 'id',
            'type' => 'varchar',
            'len'=>'36',
        ),
        // account id
        array(
            'name' => 'account_id',
            'type' => 'varchar',
            'len'=>'36'
        ),
        // bug id
        array(
            'name' => 'bug_id',
            'type' => 'varchar',
            'len' => '36'
        ),
        // necessary
        array(
            'name' => 'date_modified',
            'type' => 'datetime'
        ),
        // necessary
        array(
            'name' => 'deleted',
            'type' => 'bool',
            'len' => '1',
            'required' => true,
            'default' => '0'
        ),
    ),
```

```
    // the indices are necessary for indexing and performance
    'indices' => array (
        array(
            'name' => 'accounts_bugspk',
            'type' => 'primary',
            'fields' => array('id')
        ),
        array(
            'name' => 'idx_acc_bug_acc',
            'type' => 'index',
            'fields' => array('account_id')
        ),
        array(
            'name' => 'idx_acc_bug_bug',
            'type' => 'index',
            'fields' => array('bug_id')
        ),
        array(
            'name' => 'idx_account_bug',
            'type' => 'alternate_key',
            'fields' => array('account_id', 'bug_id')
        )
    ),
    'relationships' => array(
        'accounts_bugs' => array(
            // the left hand module - should match $beanList index
            'lhs_module' => 'Accounts',
            // the left hand table name
            'lhs_table' => 'accounts',
            // the key to use from the left table
            'lhs_key' => 'id',
            // the right hand module - should match $beanList index
            'rhs_module' => 'Bugs',
            // the right hand table name
            'rhs_table' => 'bugs',
            // the key to use from the right table
            'rhs_key' => 'id',
            // relationship type
            'relationship_type' => 'many-to-many',
            // join table - table used to join items
            'join_table' => 'accounts_bugs',
            // left table key - should exist in table field
definitions above
            'join_key_lhs' => 'account_id',
            // right table key - should exist in table field
definitions above
```

```
            'join_key_rhs' => 'bug_id'
        ),
    ),
);
```

Layout Defs

This is the file that contains the related modules to create subpanels for. It is stored in the $layout_defs array

`$layout_defs[<module>]['subpanel_setup'][<related_module>].`

This example is from the account metadata/subpaneldefs.php

```
'contacts' => array(
    // the order in which this subpanel is displayed with other
subpanels
    'order' => 30,
    'module' => 'Contacts',
    // in this case, it will use
./modules/Contacts/subpanels/default.php
    'subpanel_name' => 'default',
    'get_subpanel_data' => 'contacts',
    'add_subpanel_data' => 'contact_id',
    'title_key' => 'LBL_CONTACTS_SUBPANEL_TITLE',
    // this array defines the top buttons
    'top_buttons' => array(
        array(
            'widget_class' => 'SubPanelTopCreateAccountNameButton'
        ),
        array(
            'widget_class' => 'SubPanelTopSelectButton',
            'mode' => 'MultiSelect'
        )
    ),
),
```

More information on layoutdefs can be found in the Layoutdef Extension section.

In the language file for the module containing the subpanel, the following values need to be added.

•	The reference used in 'title_key' as

	shown above

•	In the example, it would be: `$mod_strings['LBL_CONTACTS_SUBPANEL_TITLE'] = 'Contacts';`

•	The reference used in 'vname' as shown in the vardefs section

In the example, it would be:

```
$mod_strings['LBL_CASES'] = 'Cases';
```

More information on language can be found in the Language Extensions and Language Framework sections.

Last Modified: 01/15/2016 10:13pm

Examples

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 09/26/2015 04:23pm

Dynamically Collapsing Subpanels Based on Record Values

Overview

This article will demonstrate how to dynamically collapse subpanels that are dependent on record values by overriding the DetailView.

Use Case

This example is for an account record where we will collapse the Contacts, Leads, Documents and Cases subpanels when the account type is 'Analyst'.

Example

Create the file ./custom/modules/Accounts/views/view.detail.php

```
<?php

    if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    //Points to 'modules/Accounts/views/view.detail.php' instead of
'include/MVC/View/views/view.detail.php' since accounts already
overrides the stock view
    //require_once('include/MVC/View/views/view.detail.php');
    require_once('modules/Accounts/views/view.detail.php');

    class CustomAccountsViewDetail extends AccountsViewDetail
    {

        function CustomAccountsViewDetail()
        {
            parent::AccountsViewDetail();
        }

        /**
         * Override - Called from process(). This method will display
subpanels.
         */
        protected function _displaySubPanels()
        {
            //Retrieve the subpanel cooke
            $panelSetup = array();
            $cookie = $_COOKIE[$this->bean->module_dir . '_divs'];

            if(!empty($cookie))
            {
                $subpanelCookies = explode('#', $cookie);

                foreach($subpanelCookies as $subpanelSettings)
                {
                    $setting = explode('=', $subpanelSettings);

                    if (isset($setting[1]))
                    {
                        $panelSetup[$setting[0]] = $setting[1];
                    }
```

```
                else
                {
                    $panelSetup[$setting[0]] = '';
                }
            }
        }

        //Subpanels to collapse
        $subpanelsToCollapse = array(
            'contacts',
            'documents',
            'leads',
            'cases',
        );

        //Dependent logic
        if ($this->bean->account_type == 'Analyst')
        {
            foreach ($subpanelsToCollapse as $subpanel)
            {
                $panelSetup["{$subpanel}_v"] = 'none';
            }

            $newCookie = '';
            foreach($panelSetup as $key => $value)
            {
                if (!empty($key))
                {
                    $newCookie .= "{$key}={$value}#";
                }
            }

            //Update cookie
            $_COOKIE[$this->bean->module_dir . '_divs'] =
$newCookie;
        }

        parent::_displaySubPanels();

    }
}

?>
```

Last Modified: 01/08/2017 04:38pm

Dynamically Hiding Subpanels Based on Record Values

Overview

This article will demonstrate how to dynamically hide subpanels that are dependent on record values by overriding the DetailView.

Use Case

This example is for an account record where we will hide the Contacts, Leads, Documents and Cases subpanels when the account type is 'Analyst'.

Example

Create the file ./custom/modules/Accounts/views/view.detail.php

```
<?php

    if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    //Points to 'modules/Accounts/views/view.detail.php' instead of
'include/MVC/View/views/view.detail.php' since accounts already
overrides the stock view
    //require_once('include/MVC/View/views/view.detail.php');
    require_once('modules/Accounts/views/view.detail.php');

    class CustomAccountsViewDetail extends AccountsViewDetail
    {

        function CustomAccountsViewDetail()
        {
            parent::AccountsViewDetail();
        }

        /**
         * Override - Called from process(). This method will display
subpanels.
         */
        protected function _displaySubPanels()
        {
```

```
            if (isset($this->bean) && !empty($this->bean->id) &&
(file_exists('modules/' . $this->module .
'/metadata/subpaneldefs.php') || file_exists('custom/modules/' .
$this->module . '/metadata/subpaneldefs.php') ||
file_exists('custom/modules/' . $this->module .
'/Ext/Layoutdefs/layoutdefs.ext.php'))) {
                $GLOBALS['focus'] = $this->bean;
                require_once ('include/SubPanel/SubPanelTiles.php');
                $subpanel = new SubPanelTiles($this->bean,
$this->module);

                //Dependent logic
                if ($this->bean->account_type == 'Analyst')
                {
                    //Subpanels to hide
                    $hideSubpanels=array(
                        'contacts',
                        'leads',
                        'documents',
                        'cases',
                    );

                    foreach ($hideSubpanels as $subpanelKey)
                    {
                        //Remove subpanel if set
                        if
(isset($subpanel->subpanel_definitions->layout_defs['subpanel_setup'][
$subpanelKey]))
                        {

unset($subpanel->subpanel_definitions->layout_defs['subpanel_setup'][$
subpanelKey]);
                        }
                    }
                }

                echo $subpanel->display();
            }
        }
    }

?>
```

Last Modified: 01/08/2017 04:38pm

Adding a Target Lists Subpanel to Leads and Contacts

Overview

Target lists, the list of recipients for email campaigns, often include contacts and leads. Displaying a Target Lists subpanel on the detail view of these modules allows greater visibility into marketing efforts related to the lead or contact. This article will explain how a developer can add a Target Lists subpanel to the Leads and Contacts detail view in Sugar versions 6.5 and 6.7. The subpanel will display one line item for every target list related to the parent lead or contact.

Note: These instructions are not compatible with Sugar 7.x which uses the Sidecar framework. For more information about adding subpanels to Sidecar modules, view the Extension Framework section of the Developer Guide for your version of Sugar.

Steps to Complete

Adding a Target Lists Subpanel to Leads

Use the following steps to add a Target Lists subpanel to the Leads detail view:

First, we will need to create our subpanel label extension file in ./custom/Extension/modules/Leads/Ext/Language/. You are free to choose your own file name, however, you must prefix the file with your preferred language such as en_us. and the suffix .php. For our purposes, we will create:

./custom/Extension/modules/Leads/Ext/Language/en_us.prospectlists_leads_Leads.php

```
<?php

$mod_strings['LBL_PROSPECTLISTS_LEADS_FROM_PROSPECTLISTS_TITLE'] =
'Target Lists';
```

Next, we will need to create the vardef link definition for our subpanel in ./custom/Extension/modules/Leads/Ext/Vardefs/. You are free to choose your own file name. For our purposes, we will create:

./custom/Extension/modules/Leads/Ext/Vardefs/prospectlists_leads_Leads.php

```
<?php
```

```
$dictionary["Lead"]["fields"]["prospect_list_leads"] = array (
  'name' => 'prospect_list_leads',
  'type' => 'link',
  'relationship' => 'prospect_list_leads',
  'source' => 'non-db',
  'vname' => 'LBL_PROSPECTLISTS_LEADS_FROM_PROSPECTLISTS_TITLE',
);
```

To complete our new subpanel, we will create our layout definition in ./custom/Extension/modules/Leads/Ext/Layoutdefs/. You are free to choose your own file name. For our purposes, we will create:

./custom/Extension/modules/Leads/Ext/Layoutdefs/prospectlists_leads_Leads.php

```
<?php

$layout_defs["Leads"]["subpanel_setup"]['prospect_list_leads'] = array
(
  'order' => 100,
  'module' => 'ProspectLists',
  'subpanel_name' => 'default',
  'sort_order' => 'asc',
  'sort_by' => 'id',
  'title_key' => 'LBL_PROSPECTLISTS_LEADS_FROM_PROSPECTLISTS_TITLE',
  'get_subpanel_data' => 'prospect_list_leads',
  'top_buttons' => array (
    0 => array (
    'widget_class' => 'SubPanelTopButtonQuickCreate',
    ),
    1 => array (
    'widget_class' => 'SubPanelTopSelectButton',
    'mode' => 'MultiSelect',
    ),
  ),
);
```

Once the files are created, you will need to navigate to Admin > Repair > Quick Repair and Rebuild. Once completed, the Target List subpanel will not be available under Leads.

Adding a Target Lists Subpanel to Contacts

Use the following steps to add a Target Lists subpanel to the Contacts detail view:

First, we will need to create our subpanel label extension file in ./custom/Extension/modules/Contacts/Ext/Language/. You are free to choose your own file name, however, you must prefix the file with your preferred language such as en_us. and the suffix .php. For our purposes, we will create:

./custom/Extension/modules/Contacts/Ext/Language/en_us.prospectlists_contacts_Contacts.php

```
<?php

$mod_strings['LBL_PROSPECTLISTS_CONTACTS_FROM_PROSPECTLISTS_TITLE'] = 'Target Lists';
```

Next, we will need to create the vardef link definition for our subpanel in ./custom/Extension/modules/Contacts/Ext/Vardefs/. You are free to choose your own file name. For our purposes, we will create:

./custom/Extension/modules/Contacts/Ext/Vardefs/prospectlists_contacts_Contacts.php

```
<?php

$dictionary["Contact"]["fields"]["prospect_list_contacts"] = array (
  'name' => 'prospect_list_contacts',
  'type' => 'link',
  'relationship' => 'prospect_list_contacts',
  'source' => 'non-db',
  'vname' => 'LBL_PROSPECTLISTS_CONTACTS_FROM_PROSPECTLISTS_TITLE',
);
```

To complete our new subpanel, we will create our layout definition in ./custom/Extension/modules/Contacts/Ext/Layoutdefs/. You are free to choose your own file name. For our purposes, we will create:

./custom/Extension/modules/Contacts/Ext/Layoutdefs/prospectlists_contacts_Contacts.php

```
<?php

$layout_defs["Contacts"]["subpanel_setup"]['prospect_list_contacts'] = array (
  'order' => 100,
  'module' => 'ProspectLists',
  'subpanel_name' => 'default',
  'sort_order' => 'asc',
  'sort_by' => 'id',
```

```
  'title_key' =>
'LBL_PROSPECTLISTS_CONTACTS_FROM_PROSPECTLISTS_TITLE',
  'get_subpanel_data' => 'prospect_list_contacts',
  'top_buttons' => array (
    0 => array (
    'widget_class' => 'SubPanelTopButtonQuickCreate',
    ),
    1 => array (
    'widget_class' => 'SubPanelTopSelectButton',
    'mode' => 'MultiSelect',
    ),
  ),
);
```

Once the files are created, you will need to navigate to Admin > Repair > Quick Repair and Rebuild. Once completed, the Target List subpanel will be available under Contacts.

Last Modified: 10/04/2016 02:38pm

Making Activity Creation Open in Full Form

Making Activities Open in Full Form

Overview

Many Sugar modules contain an Activities subpanel that allows for quickly creating tasks, calls, and meetings. Creating these activity records from the subpanel opens an abbreviated version of an edit view called the quick create view. Quick create often includes fewer fields than the full edit view. Depending on your business needs, users may prefer to use the full form edit view when creating a task, call, or meeting through the subpanel. This article will show you how to disable the Quick Create feature for the Activities subpanel in all modules, for version 6.7.x and prior.

Steps to Complete

To achieve our customization, we will extend the stock SugarWidgetSubPanelTopButton widget for each of the activity modules. Once this has been done, our new buttons will be loaded in place of the stock quickcreate buttons. First, we will need to override the Tasks button in the Activities subpanel. To accomplish this, we will create:

custom/include/generic/SugarWidgets/SugarWidgetSubPanelTopCreateTaskButton.php

```
<?php

class SugarWidgetSubPanelTopCreateTaskButton extends
SugarWidgetSubPanelTopButton
{
    var $module = 'Tasks';
    var $title = 'Create Task';
    var $access_key;
    var $form_value;
    var $additional_form_fields;
    var $acl;
}
```

custom/include/generic/SugarWidgets/SugarWidgetSubPanelTopScheduleCallButton.php

```
<?php

class SugarWidgetSubPanelTopScheduleCallButton extends
SugarWidgetSubPanelTopButton{
  var $module = 'Calls';
  var $title = 'Log Call';
  var $access_key;
  var $form_value;
  var $additional_form_fields;
  var $acl;
}
```

We will then override the Meetings button in the Activities subpanel. To accomplish this, we will create:

custom/include/generic/SugarWidgets/SugarWidgetSubPanelTopScheduleMeetingButton.php

```
<?php

class SugarWidgetSubPanelTopScheduleMeetingButton extends
SugarWidgetSubPanelTopButton{
    var $module = 'Meetings';
    var $title = 'Schedule Meeting';
    var $access_key;
    var $form_value;
    var $additional_form_fields;
```

```
    var $acl;
}
```

Once all of the files are in place, we will need to navigate to Admin > Repair > Quick Repair and Rebuild.

Summary

Once the above steps are completed, the full edit view will automatically appear when users create records from the Activities subpanel on any module. This will allow users to create more detailed tasks, calls, and meetings as more fields will be visible during creation.

Last Modified: 01/08/2017 04:38pm

Logic Hooks

Logic Hooks.

Last Modified: 11/19/2015 03:22am

Introduction

Logic Hooks

The Logic Hook framework allows you to add actions to specific events that occur within the system.

Hook Definitions

The logic hook actions and trigger events are defined in the hook definition.

Definition Locations

Custom logic hook definitions can be located in the following paths:

Module Specific Paths	
./custom/Extension/modules/<module>/Ext/LogicHooks/<file>.php	Defines hook actions that will be executed for the specified events on a specific module using the extension framework. This path should be used when developing module specific hooks.
./custom/modules/<module>/logic_hooks.php	Defines hook actions that will be executed for the specified events on a specific module. You should avoid using this path as it may be overwritten by module loadable plugins.
Application Paths	
./custom/Extension/application/Ext/LogicHooks/<file>.php	Defines hook actions that will be executed for the specified events on all modules using the extension framework. This path should be used when developing application hooks.
./custom/modules/logic_hooks.php	Defines hook actions that will be executed for the specified events on all modules. You should avoid using this path as it may be overwritten by module loadable plugins.

Definition Properties

The logic hooks must have $hook_version and $hook_array defined in its definition before it can be used by the system.

hook_version

All logic hook definitions will define a $hook_version. This determines the version of the logic hook framework. Currently, the only supported hook version is 1.

```
$hook_version = 1;
```

hook_array

The logic hook definition will also contain a $hook_array. This defines the specific of the action to execute. The hook array will be defined as follows:

Name	Type	Description

event_name	String	The name of the event that append the action to.
process_index	Integer	The order to execute the actions in.
description	String	A short description to identify the hook action.
file_path	String	The path to the logic hook file. The file should reside within the ./custom/ directory.
class_name	String	The name of the logic hook action class.
method_name	String	The name of the logic hook action method.

```
$hook_array['<event_name>'][] = array(
    <process_index>, //Integer
    '<description>', //String
    '<file_path>', //String
    '<class_name>', //String
    '<method_name>', //String
);
```

Example Definition

```
<?php

    $hook_array['before_save'][] = array(
        1,
        'Hook description',
        'custom/modules/Accounts/customLogicHook.php',
        'className',
        'methodName'
    );
?>
```

Hook Action Method

The hook action class can be located anywhere you choose. For best practices, it is recommended to group the hooks with the module they are associated with within the ./custom/ directory. An example of a method class for the definition above is shown below:

```
<?php
```

```
    class className
    {
        function methodName($bean, $event, $arguments)
        {
            //logic
        }
    }
?>
```

The logic hook method signature may contain different arguments depending on the hook event.

Considerations

A few cautions around using logic hooks:

- As of PHP 5.3, objects are automatically passed by reference. When creating your logic hook signatures, it is important that you do not append the ampersand (&) to the $bean variable. Doing this will cause unexpected behavior.
- There is no hook that fires specifically for the ListView, DetailView or EditView events. You will need to use either the 'process_record' or 'after_retrieve' logic hooks.
- In order to compare new values with previous values to determine whether a change has happened, you can use fetched_row to check. An Example is below:

```
if ($bean->fetched_row['{field}'] != $bean->{field})
{
    //logic
}
```

- Make sure that the permissions on your logic_hooks.php file and the class file that it references are readable by the web server. If this is not done, Sugar will not read the files and your business logic extensions will not work. For example, *nix developers who are extending the Tasks logic should use the following command for the logic_hooks file and the same command for the class file that will be called.

```
chmod +r ./custom/modules/Tasks/logic_hooks.php
```

- Make sure that the entire ./custom/ directory is writable by the web server or else the logic hooks code will not work properly.

Last Modified: 01/15/2016 10:09pm

Application Hooks

Logic Hooks that can be used to execute logic when working with the global application.

Logic Hooks that can be used to execute logic when working with the global application.

Last Modified: 11/19/2015 04:52am

after_entry_point

Overview

Executes at the start of every request.

Definition

```
function after_entry_point($event, $arguments){}
```

Arguments

Name	Type	Description
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Considerations

- This is a global logic hook where the logic hook reference must be placed in ./custom/modules/logic_hooks.php.

- This hook is executed at the start of every request at the end of ./include/entryPoint.php.
- This hook should not be used for any type of display output.
- Ideally used for logging or loading libraries.
- Application hooks do not make use of the $bean argument.

Change Log

Version	Note
6.4.3	Added after_entry_point hook.

Example

./custom/modules/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['after_entry_point'] = Array();
    $hook_array['after_entry_point'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'after_entry_point example',

        //The PHP file where your class is located.
        'custom/modules/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'after_entry_point_method'
    );

?>
```

./custom/modules/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function after_entry_point_method($event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:05pm

after_ui_footer

Overview

Executes after the footer has been invoked.

Definition

```
function after_ui_footer($event, $arguments){}
```

Arguments

Name	Type	Description
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Considerations

- This is a global logic hook where the logic hook reference must be placed in ./custom/modules/logic_hooks.php.
- If you are intending to write display logic to the screen, you must first wrap the display logic in an if condition to prevent breaking the Ajax page loads. This logic may vary and it is best to only run your code on specific pages rather than all pages to avoid issues. An example of this is shown below:

```
if (!isset($_REQUEST["to_pdf"]) || $_REQUEST["to_pdf"] == false)
{
    //display logic
}
```

- This hook is executed on all page loads.
- Application hooks do not make use of the $bean argument.

Change Log

Version	Note
5.0.0a	Added after_ui_footer hook.

Example

./custom/modules/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['after_ui_footer'] = Array();
    $hook_array['after_ui_footer'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'after_ui_footer example',

        //The PHP file where your class is located.
        'custom/modules/logic_hooks_class.php',

        //The class the method is in.
```

```
        'logic_hooks_class',

        //The method to call.
        'after_ui_footer_method'
    );

?>
```

./custom/modules/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function after_ui_footer_method($event, $arguments)
        {
            //display logic should check for $_REQUEST["to_pdf"]
        }
    }

?>
```

Last Modified: 01/15/2016 10:05pm

after_ui_frame

Overview

Executes after the frame has been invoked and before the footer has been invoked.

Definition

```
function after_ui_frame($event, $arguments){}
```

Arguments

Name	Type	Description

Name	Type	Description
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Considerations

- This hook is executed on most views such as the DetailView, EditView, and Listview.
- Application hooks do not make use of the $bean argument.
- This is logic hook can be used as a global reference (./custom/modules/logic_hooks.php) or as a module reference (./custom/modules/<module>/logic_hooks.php).

Change Log

Version	Note
5.0.0a	Added after_ui_frame hook.

Example

./custom/modules/{module}/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['after_ui_frame'] = Array();
    $hook_array['after_ui_frame'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'after_ui_frame example',
```

```
        //The PHP file where your class is located.
        'custom/modules/{module}/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'after_ui_frame_method'
    );

?>
```

./custom/modules/{module}/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function after_ui_frame_method($event, $arguments)
        {
            //display logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:05pm

handle_exception

Overview

Executes when an exception is thrown.

Definition

```
function handle_exception($event, $exception){}
```

Arguments

Name	Type	Description
event	String	The current event.
exception	Object	The exception object.

Considerations

- This hook should not be used for any type of display output.
- You can retrieve the exception message by using $exception->getMessage(). All exception classes extend the PHP Exceptions class.

Change Log

Version	Note
6.4.4	Added handle_exception hook.

Example

./custom/modules/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['handle_exception'] = Array();
    $hook_array['handle_exception'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'handle_exception example',

        //The PHP file where your class is located.
        'custom/modules/logic_hooks_class.php',
```

```
        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'handle_exception_method'
    );

?>
```

./custom/modules/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function handle_exception_method($event, $exception)
        {
            //logic with $exception->getMessage()
        }
    }

?>
```

Last Modified: 11/28/2016 11:50am

server_round_trip

Overview

Executes at the end of every page.

Definition

```
function server_round_trip($event, $arguments){}
```

Arguments

Name	Type	Description
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Considerations

- This is a global logic hook where the logic hook reference must be placed in ./custom/modules/logic_hooks.php.
- If you are intending to write display logic to the screen, you must first wrap the display logic in an if condition to prevent breaking the Ajax page loads:

```
if (!isset($_REQUEST["to_pdf"]) || $_REQUEST["to_pdf"] == false)
{
    //display logic
}
```

- This hook is executed on all page loads.
- Application hooks do not make use of the $bean argument.

Change Log

Version	Note
5.0.0a	Added server_round_trip hook.

Example

./custom/modules/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['server_round_trip'] = Array();
    $hook_array['server_round_trip'][] = Array(
```

```
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'server_round_trip example',

        //The PHP file where your class is located.
        'custom/modules/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'server_round_trip_method'
    );

?>
```

./custom/modules/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function server_round_trip_method($event, $arguments)
        {
            //display logic should check for $_REQUEST["to_pdf"]
        }
    }

?>
```

Last Modified: 01/15/2016 10:05pm

Module Hooks

Logic Hooks that can be used to execute logic when working with modules.

Last Modified: 09/26/2015 04:23pm

after_delete

Overview

Executes after a record is deleted.

Definition

```
function after_delete($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event.
arguments.id	String	Id of the deleted record.

Example

./custom/modules/{module}/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['after_delete'] = Array();
    $hook_array['after_delete'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'after_delete example',

        //The PHP file where your class is located.
        'custom/modules/{module}/logic_hooks_class.php',
```

```
        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'after_delete_method'
    );

?>
```

./custom/modules/{module}/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function after_delete_method($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:09pm

after_relationship_add

Overview

Executes after a relationship has been added between two records.

Definition

```
function after_relationship_add($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event.
arguments.id	String	Module id.
arguments.module	String	Module name.
arguments.related_id	String	Related module id.
arguments.related_module	String	Related module name.
arguments.link	String	Link field name.
arguments.relationship	String	Relationship name.

Considerations

- This hook will be executed for each side of the relationship. An example is that if you add an association between an Account and Contact, the hook will be run for both.
- The arguments parameter will have additional information regarding the records being modified. The $bean variable will not contain this information.

Change Log

Version	Note
6.0.0	Added after_relationship_add hook.

Example

./custom/modules/{module}/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['after_relationship_add'] = Array();
```

```
    $hook_array['after_relationship_add'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'after_relationship_add example',

        //The PHP file where your class is located.
        'custom/modules/{module}/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'after_relationship_add_method'
    );

?>
```

./custom/modules/{module}/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function after_relationship_add_method($bean, $event,
$arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:09pm

after_relationship_delete

Overview

Executes after a relationship has been deleted between two records.

Definition

```
function after_relationship_delete($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event.
arguments.id	String	Module id.
arguments.module	String	Module name.
arguments.related_id	String	Related module id.
arguments.related_module	String	Related module name.
arguments.link	String	Link field name.
arguments.relationship	String	Relationship name.

Considerations

- This hook will be executed for each side of the relationship. An example is that if you delete an association between an Account and Contact, the hook will be run for both.
- The arguments parameter will have additional information regarding the records being modified. The $bean variable will not contain this information.

Change Log

Version	Note
6.0.0	Added after_relationship_delete hook.

Example

./custom/modules/{module}/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['after_relationship_delete'] = Array();
    $hook_array['after_relationship_delete'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'after_relationship_delete example',

        //The PHP file where your class is located.
        'custom/modules/{module}/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'after_relationship_delete_method'
    );

?>
```

./custom/modules/{module}/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function after_relationship_delete_method($bean, $event,
$arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:13pm

after_restore

Overview

Executes after a record is undeleted.

Definition

```
function after_restore($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Example

./custom/modules/{module}/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['after_restore'] = Array();
    $hook_array['after_restore'][] = Array(
        //Processing index. For sorting the array.
        1,
```

```
        //Label. A string value to identify the hook.
        'after_restore example',

        //The PHP file where your class is located.
        'custom/modules/{module}/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'after_restore_method'
    );

?>
```

./custom/modules/{module}/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function after_restore_method($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:13pm

after_retrieve

Overview

Executes after a record has been retrieved from the database.

Definition

```
function after_retrieve($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event.
arguments.id	String	Id of the record.
arguments.encode	Boolean	Whether or not to encode.

Considerations

- This hook does not fire when a new record is being created.
- Calling save on the bean in this hook can cause adverse side effects if not handled correctly. (i.e: $bean->save())

Example

./custom/modules/{module}/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['after_retrieve'] = Array();
    $hook_array['after_retrieve'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'after_retrieve example',

        //The PHP file where your class is located.
        'custom/modules/{module}/logic_hooks_class.php',
```

```
        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'after_retrieve_method'
    );

?>
```

./custom/modules/{module}/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function after_retrieve_method($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:21pm

after_save

Overview

Executes after a record is saved.

Definition

```
function after_save($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Considerations

- Calling save on the bean in this hook will cause an infinite loop if not handled correctly. (i.e: $bean->save())

Change Log

Version	Note
4.5.0c	Added after_save hook.

Example

./custom/modules/{module}/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['after_save'] = Array();
    $hook_array['after_save'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'after_save example',

        //The PHP file where your class is located.
        'custom/modules/{module}/logic_hooks_class.php',
```

```
        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'after_save_method'
    );

?>
```

./custom/modules/{module}/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function after_save_method($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:25pm

before_delete

Overview

Executes before a record is deleted.

Definition

```
function before_delete($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event.
arguments.id	String	Id of the record to delete.

Example

./custom/modules/{module}/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['before_delete'] = Array();
    $hook_array['before_delete'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'before_delete example',

        //The PHP file where your class is located.
        'custom/modules/{module}/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'before_delete_method'
    );

?>
```

./custom/modules/{module}/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
```

```
    class logic_hooks_class
    {
        function before_delete_method($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:09pm

before_relationship_add

Overview

Executes before a relationship has been added between two records.

Definition

```
function before_relationship_add($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event.
arguments.id	String	Module id.
arguments.module	String	Module name.
arguments.related_id	String	Related module id.
arguments.related_module	String	Related module name.
arguments.link	String	Link field name.
arguments.relationship	String	Relationship name.

Considerations

- This hook will be executed for each side of the relationship. An example is that if you add an association between an Account and Contact, the hook will be run for both.
- The arguments parameter will have additional information regarding the records being modified. The $bean variable will not contain this information.

Change Log

Version	Note
6.4.5	Added before_relationship_add hook.

Example

./custom/modules/{module}/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['before_relationship_add'] = Array();
    $hook_array['before_relationship_add'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'before_relationship_add example',

        //The PHP file where your class is located.
        'custom/modules/{module}/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'before_relationship_add_method'
    );
```

```
?>
```

./custom/modules/{module}/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function before_relationship_add($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:09pm

before_relationship_delete

Overview

Executes before a relationship has been added between two records.

Definition

```
function before_relationship_delete($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event.

Name	Type	Description
arguments.id	String	Module id.
arguments.module	String	Module name.
arguments.related_id	String	Related module id.
arguments.related_module	String	Related module name.
arguments.link	String	Link field name.
arguments.relationship	String	Relationship name.

Considerations

- This hook will be executed for each side of the relationship. An example is that if you add an association between an Account and Contact, the hook will be run for both.
- The arguments parameter will have additional information regarding the records being modified. The $bean variable will not contain this information.

Change Log

Version	Note
6.4.5	Added before_relationship_delete hook.

Example

./custom/modules/{module}/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['before_relationship_delete'] = Array();
    $hook_array['before_relationship_delete'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'before_relationship_delete example',
```

```
        //The PHP file where your class is located.
        'custom/modules/{module}/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'before_relationship_delete_method'
    );

?>
```

./custom/modules/{module}/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function before_relationship_delete_method($bean, $event,
$arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:09pm

before_restore

Overview

Executes before a record is undeleted.

Definition

```
function before_restore($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Example

./custom/modules/{module}/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['before_restore'] = Array();
    $hook_array['before_restore'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'before_restore example',

        //The PHP file where your class is located.
        'custom/modules/{module}/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'before_restore_method'
    );

?>
```

./custom/modules/{module}/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function before_restore_method($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:09pm

before_save

Overview

Executes before a record is saved.

Definition

```
function before_save($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event.
arguments.check_notify	Boolean	Whether or not to send notifications.

Considerations

- With certain modules, like Cases and Bugs, the human-readable ID of the record (like the case_number field in the Case module), is not available within a before_save call. This is because this business logic has not been executed yet.
- Calling save on the bean in this hook will cause an infinite loop if not handled correctly. (i.e: $bean->save())

Example

./custom/modules/{module}/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['before_save'] = Array();
    $hook_array['before_save'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'before_save example',

        //The PHP file where your class is located.
        'custom/modules/{module}/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'before_save_method'
    );

?>
```

./custom/modules/{module}/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
```

```
    class logic_hooks_class
    {
        function before_save_method($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:25pm

process_record

Overview

Executes when the record is being processed as a part of the ListView or subpanel list.

Definition

```
function process_record($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Considerations

- This can be used to set values in a record's fields prior to display in the

ListView or in the subpanel of a DetailView.
- This event is not fired in the EditView.

Change Log

Version	Note
5.0.0a	Added process_record hook.

Example

./custom/modules/{module}/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['process_record'] = Array();
    $hook_array['process_record'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'process_record example',

        //The PHP file where your class is located.
        'custom/modules/{module}/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'process_record_method'
    );

?>
```

./custom/modules/{module}/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
```

```
Point');

    class logic_hooks_class
    {
        function process_record_method($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:29pm

User Hooks

Logic Hooks that can be used to execute logic when working with the Users module.

Last Modified: 09/26/2015 04:23pm

after_load_user

Overview

Executes after the current user is set for the current request. Used to handle actions that are dependent on the current current user such as setting ACLs or configuring user-dependent parameters.

Definition

```
function after_load_user($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.

Name	Type	Description
arguments	Array	Additional information related to the event. This parameter is normally empty.

Change Log

Version	Note
6.6.0	Added after_load_user hook.

Example

./custom/modules/Users/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['after_load_user'] = Array();
    $hook_array['after_load_user'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'after_load_user example',

        //The PHP file where your class is located.
        'custom/modules/Users/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'after_load_user_method'
    );

?>
```

./custom/modules/Users/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function after_load_user($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:29pm

after_login

Overview

Executes after a user logs into the system.

Definition

```
function after_login($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Change Log

Version	Note
5.0.0a	Added after_login hook.

Example

./custom/modules/Users/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['after_login'] = Array();
    $hook_array['after_login'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'after_login example',

        //The PHP file where your class is located.
        'custom/modules/Users/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'after_login_method'
    );

?>
```

./custom/modules/Users/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
```

```
    class logic_hooks_class
    {
        function after_login_method($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:13pm

after_logout

Overview

Executes after a user logs out of the system.

Definition

```
function after_logout($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Change Log

Version	Note
5.0.0a	Added after_logout hook.

Version	Note

Example

./custom/modules/Users/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['after_logout'] = Array();
    $hook_array['after_logout'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'after_logout example',

        //The PHP file where your class is located.
        'custom/modules/Users/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'after_logout_method'
    );

?>
```

./custom/modules/Users/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function after_logout_method($bean, $event, $arguments)
        {
            //logic
        }
```

```
    }
?>
```

Last Modified: 01/15/2016 10:13pm

before_logout

Overview

Executes before a user logs out of the system.

Definition

```
function before_logout($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Change Log

Version	Note
5.0.0a	Added before_logout hook.

Example

./custom/modules/Users/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['before_logout'] = Array();
    $hook_array['before_logout'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'before_logout example',

        //The PHP file where your class is located.
        'custom/modules/Users/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'before_logout_method'
    );

?>
```

./custom/modules/Users/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function before_logout_method($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:13pm

login_failed

Overview

Executes on a failed login attempt.

Definition

```
function login_failed($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The bean object.
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Change Log

Version	Note
5.0.0a	Added login_failed hook.

Example

./custom/modules/Users/logic_hooks.php

```
<?php
```

```
    $hook_version = 1;
    $hook_array = Array();

    $hook_array['login_failed'] = Array();
    $hook_array['login_failed'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'login_failed example',

        //The PHP file where your class is located.
        'custom/modules/Users/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'login_failed_method'
    );

?>
```

./custom/modules/Users/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function login_failed_method($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:13pm

Job Queue Hooks

Logic Hooks that can be used to execute logic when working with the Job Queue module.

Last Modified: 09/26/2015 04:23pm

job_failure

Overview

Executes when a jobs final failure occurs.

Definition

```
function job_failure($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The SchedulersJob object.
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Change Log

Version	Note
6.5.0RC1	Added job_failure hook.

Example

./custom/modules/SchedulersJobs/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['job_failure'] = Array();
    $hook_array['job_failure'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'job_failure example',

        //The PHP file where your class is located.
        'custom/modules/SchedulersJobs/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'job_failure_method'
    );

?>
```

./custom/modules/SchedulersJobs/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function job_failure_method($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:29pm

job_failure_retry

Overview

Executes any time a job fails before its final failure. Use job_failure if you only want action on a final failure.

Definition

```
function job_failure_retry($bean, $event, $arguments){}
```

Arguments

Name	Type	Description
bean	Object	The SchedulersJob object.
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Change Log

Version	Note
6.5.0RC1	Added job_failure_retry hook.

Example

./custom/modules/SchedulersJobs/logic_hooks.php

```
<?php

    $hook_version = 1;
```

```
    $hook_array = Array();

    $hook_array['job_failure_retry'] = Array();
    $hook_array['job_failure_retry'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'job_failure_retry example',

        //The PHP file where your class is located.
        'custom/modules/SchedulersJobs/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'job_failure_retry_method'
    );
?>
```

./custom/modules/SchedulersJobs/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function job_failure_retry_method($bean, $event, $arguments)
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:17pm

SNIP Hooks

Logic Hooks that can be used to execute logic when working with SNIP.

Last Modified: 09/26/2015 04:23pm

after_email_import

Overview

Executes after a SNIP email has been imported.

Definition

```
function after_email_import($bean, $event, $arguments){}
```

Parameters

Name	Type	Description
bean	Object	The Email object.
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Considerations

- This is a global logic hook where the logic hook reference must be placed in ./custom/modules/logic_hooks.php.

Change Log

Version	Note
6.5.0	Added after_email_import hook.

Example

./custom/modules/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['after_email_import'] = Array();
    $hook_array['after_email_import'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'after_email_import example',

        //The PHP file where your class is located.
        'custom/modules/SNIP/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'after_email_import_method'
    );

?>
```

./custom/modules/SNIP/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function after_email_import_method($bean, $event, $arguments)
        {
            //logic
        }
    }
```

?>

Last Modified: 01/15/2016 10:09pm

before_email_import

Overview

Executes before a SNIP email has been imported.

Definition

```
function before_email_import($bean, $event, $arguments){}
```

Parameters

Name	Type	Description
bean	Object	The Email object.
event	String	The current event.
arguments	Array	Additional information related to the event. This parameter is normally empty.

Considerations

- This is a global logic hook where the logic hook reference must be placed in ./custom/modules/logic_hooks.php.

Change Log

Version	Note

Version	Note
6.5.0	Added before_email_import hook.

Example

./custom/modules/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();

    $hook_array['before_email_import'] = Array();
    $hook_array['before_email_import'][] = Array(
        //Processing index. For sorting the array.
        1,

        //Label. A string value to identify the hook.
        'before_email_import example',

        //The PHP file where your class is located.
        'custom/modules/SNIP/logic_hooks_class.php',

        //The class the method is in.
        'logic_hooks_class',

        //The method to call.
        'before_email_import_method'
    );

?>
```

./custom/modules/SNIP/logic_hooks_class.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class logic_hooks_class
    {
        function before_email_import_method($bean, $event, $arguments)
```

```
        {
            //logic
        }
    }

?>
```

Last Modified: 01/15/2016 10:09pm

Logic Hook Release Notes

Overview

Lists changes to the logic hook libraries.

Release Notes

6.6.0

- after_load_user hook was added

6.5.0RC1

- after_email_import hook was added
- before_email_import hook was added
- job_failure hook was added
- job_failure_retry hook was added

6.4.5

- before_relationship_add hook was added
- before_relationship_delete hook was added

6.4.4

- handle_exception hook was added

6.4.3

- after_entry_point hook was added

6.0.0RC1

- after_relationship_add hook was added
- after_relationship_delete hook was added

5.0.0a

- after_login hook was added
- after_logout hook was added
- after_ui_footer hook was added
- after_ui_frame hook was added
- before_logout hook was added
- login_failed hook was added
- process_record hook was added
- server_round_trip hook was added

4.5.0c

- after_save hook was added

Last Modified: 11/19/2015 03:22am

Examples

Examples of working with logic hooks.

Last Modified: 11/19/2015 12:49pm

Comparing Bean Properties Between Logic Hooks

Overview

This example will demonstrate how to compare the properties of a bean between the before_save and after_save logic hooks.

Storing and Comparing Values

When working with a bean in the after_save logic hook, you may notice that the after_save fetched rows no longer match the before_save fetched rows. If your after_save logic needs to be able to compare values that were in the before_save, you can use the following example to help you store and use the values.

Example

./custom/modules/<module>/logic_hooks.php

```
<?php

    $hook_version = 1;
    $hook_array = Array();
    $hook_array['before_save'] = Array();
    $hook_array['before_save'][] = Array(
        1,
        'Store values',
        'custom/modules/<module>/My_Logic_Hooks.php',
        'My_Logic_Hooks',
        'before_save_method'
    );

    $hook_array['after_save'] = Array();
    $hook_array['after_save'][] = Array(
        1,
        'Retrieve and compare values',
        'custom/modules/<module>/My_Logic_Hooks.php',
        'My_Logic_Hooks',
        'after_save_method'
    );

?>
```

./custom/modules/<module>/My_Logic_Hooks.php

```
<?php

if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

class My_Logic_Hooks
{
    function before_save_method($bean, $event, $arguments)
    {
        //store as a new bean property
        $bean->stored_fetched_row_c = $bean->fetched_row;
    }

    function after_save_method($bean, $event, $arguments)
    {
        //check if a fields value has changed
        if (
            isset($bean->stored_fetched_row_c)
            && $bean->stored_fetched_row_c['field'] != $bean->field
           )

        {
            //execute logic
        }
    }
}

?>
```

Last Modified: 09/26/2015 04:23pm

Manipulating Logic Hook References

Overview

How to add and remove logic hook references with code.

Logic Hooks

Adding Logic Hooks

In order to add a logic hook reference to ./custom/modules/<module>/logic_hooks.php, you can use check_logic_hook_file:

```
//Adding a logic hook
require_once("include/utils.php");

$my_hook = Array(
    999,
    'Example Logic Hook',
    'custom/modules/<module>/my_hook.php',
    'my_hook_class',
    'my_hook_function'
);

check_logic_hook_file("<module>", "before_save", $my_hook);
```

Removing Logic Hooks

Removing a logic hook reference can be done by using remove_logic_hook:

```
//Removing a logic hook
require_once("include/utils.php");

$my_hook = Array(
    999,
    'Example Logic Hook',
    'custom/modules/<module>/my_hook.php',
    'my_hook_class',
    'my_hook_function'
);

remove_logic_hook("<module>", "before_save", $my_hook);
```

Last Modified: 09/26/2015 04:23pm

Preventing Infinite Loops with Logic Hooks

Overview

Infinite loops often happen when a logic hook calls save on a bean in a scenario that triggers the same hook again.

Saving in an After Save Hook

Infinite loops can sometimes happen when you have a need to update a record after it has been run through the workflow process in the after_save hook. An example of a looping hook is shown below.

./custom/modules/Accounts/logic_hooks.php

```
<?php

$hook_version = 1;
$hook_array = Array();
$hook_array['after_save'] = Array();
$hook_array['after_save'][] = Array(
    1,
    'Update Account Record',
    'custom/modules/Accounts/Accounts_Hook.php',
    'Accounts_Hook',
    'update_self'
);
```

./custom/modules/Accounts/Accounts_Hook.php

```
<?php

if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

class Accounts_Hook
{
    function update_self($bean, $event, $arguments)
    {
        //generic condition
        if ($bean->type == 'Analyst')
        {
            //update
            $bean->industry = 'Banking';
            $bean->save();
        }
    }
```

```
}
```

In the example above, we have a generic condition that when met, will trigger the update of a field on the record. This will cause an infinite loop as calling the save() method will once again trigger the before_save hook and then the after_save hook again. The solution to this problem is to add a a new property on the bean to ignore any unneeded saves. In our example we will name this property "ignore_update_c" and add a check to our logic hook to eliminate the perpetual loop. An example of this is shown below:

./custom/modules/Accounts/Accounts_Hook.php

```
<?php

if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

class Accounts_Hook
{
    function update_self($bean, $event, $arguments)
    {
        //loop prevention check
        if (!isset($bean->ignore_update_c) || $bean->ignore_update_c
=== false)
        {
            //generic condition
            if ($bean->type == 'Analyst')
            {
                //update
                $bean->ignore_update_c = true;
                $bean->industry = 'Banking';
                $bean->save();
            }
        }
    }
}
```

Related Record Save Loops

Sometimes logic hooks on two separate but related modules can cause an infinite loop. The problem arises when the two modules have logic hooks that update one another. This is often done when wanting to keep a field in sync between two modules. The example below will demonstrate this behavior on the accounts /

contacts relationship:

./custom/modules/Accounts/logic_hooks.php

```
<?php

$hook_version = 1;
$hook_array = Array();
$hook_array['before_save'] = Array();
$hook_array['before_save'][] = Array(
    1,
    'Handling associated Contacts records',
    'custom/modules/Accounts/Accounts_Hook.php',
    'Accounts_Hook',
    'update_contacts'
);
```

./custom/modules/Accounts/Accounts_Hook.php

```
<?php

if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

class Accounts_Hook
{
    function update_contacts($bean, $event, $arguments)
    {
        //if relationship is loaded
        if ($bean->load_relationship('contacts'))
        {
            //fetch related beans
            $relatedContacts = $bean->contacts->getBeans();

            foreach ($relatedContacts as $relatedContact)
            {
                //perform any other contact logic
                $relatedContact->save();
            }
        }
    }
}
```

The above example will loop through all contacts related to the account and save them. At this point, the save will then trigger our contacts logic hook shown below:

./custom/modules/Contacts/logic_hooks.php

```
<?php

$hook_version = 1;
$hook_array = Array();
$hook_array['before_save'] = Array();
$hook_array['before_save'][] = Array(
    1,
    'Handling associated Accounts records',
    'custom/modules/Contacts/Contacts_Hook.php',
    'Contacts_Hook',
    'update_account'
);
```

./custom/modules/Contacts/Contacts_Hook.php

```
<?php

if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

class Contacts_Hook
{
    function update_account($bean, $event, $arguments)
    {

        //if relationship is loaded
        if ($bean->load_relationship('accounts'))
        {
            //fetch related beans
            $relatedAccounts = $bean->accounts->getBeans();

            $parentAccount = false;
            if (!empty($relatedAccounts))
            {
                //order the results
                reset($relatedAccounts);

                //first record in the list is the parent
                $parentAccount = current($relatedAccounts);
```

```
            }

            if ($parentAccount !== false && is_object($parentAccount))
            {
                //perform any other account logic
                $parentAccount->save();
            }
        }
    }
}
```

These two logic hooks will continuously trigger one another in this scenario until you run into a max_execution or memory_limit error. The solution to this problem is to add a a new property on the bean to ignore any unneeded saves. In our example we will name this property "ignore_update_c" and add a check to our logic hook to eliminate the perpetual loop. The following code snippets are copies of the two classes with the ignore_update_c property added.

./custom/modules/Accounts/Accounts_Hook.php

```
<?php

if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

class Accounts_Hook
{
    function update_contacts($bean, $event, $arguments)
    {
        //if relationship is loaded
        if ($bean->load_relationship('contacts'))
        {
            //fetch related beans
            $relatedContacts = $bean->contacts->getBeans();

            foreach ($relatedContacts as $relatedContact)
            {
                //check if the bean's ignore_update_c attribute is not
set
                if (!isset($bean->ignore_update_c) ||
$bean->ignore_update_c === false)
                {
                    //set the ignore update attribute
```

```
                    $relatedContact->ignore_update_c = true;

                    //perform any other contact logic
                    $relatedContact->save();
                }
            }
        }
    }
}
```

./custom/modules/Contacts/Contacts_Hook.php

```
<?php

if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

class Contacts_Hook
{
    function update_account($bean, $event, $arguments)
    {
        //if relationship is loaded
        if ($bean->load_relationship('accounts'))
        {
            //fetch related beans
            $relatedAccounts = $bean->accounts->getBeans();

            $parentAccount = false;
            if (!empty($relatedAccounts))
            {
                //order the results
                reset($relatedAccounts);

                //first record in the list is the parent
                $parentAccount = current($relatedAccounts);
            }

            if ($parentAccount !== false && is_object($parentAccount))
            {
                //check if the bean's ignore_update_c element is set
                if (!isset($bean->ignore_update_c) ||
$bean->ignore_update_c === false)
                {
                    //set the ignore update attribute
```

```
                    $parentAccount->ignore_update_c = true;

                    //perform any other account logic
                    $parentAccount->save();
                }
            }
        }
    }
}
```

Last Modified: 01/08/2017 05:40pm

Sugar Logic

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 04:22am

Introduction

Overview

Sugar Logic, a new feature in Sugar Enterprise and Sugar Professional, is designed to allow custom business logic that is easy to create, manage, and reuse on both the server and client.

Sugar Logic is made up of multiple components which build off each other and is extensible at every step. The base component is the Sugar Formula Engine which parses and evaluates human readable formulas. Dependencies are units made up of triggers and actions that can express custom business logic. Each dependency defines a list of actions to be performed depending on the outcome of a trigger formula.

Terminology

- Formula : An expression that conforms to the Formula Engine syntax consisting of nested functions and field variables.
- Function : A method which can be called in a formula.
- Trigger : A Formula which evaluates to either true or false. Triggers are evaluated whenever a field in the equation is updated or when a record is retrieved/saved.
- Action : A method which modifies the current record or layout in some way.
- Dependency : A complete logical unit which includes a trigger and one or more actions.

Sugar Formula Engine

Formulas

The fundamental object is called a Formula. A Formula can be evaluated for a given record using the Sugar Logic parser.

Some example formulas are:

Basic addition:

```
add(1, 2)
```

Boolean values:

```
not(equal($billing_state, "CA"))
```

Calculation:

```
multiply(number($employees), $seat_cost, 0.0833)
```

Types

Sugar Logic has several fundamental types. They are: number, string, boolean, and enum (lists). Functions may take in any of these type or combinations thereof and return as output one of these types. Fields may also often have their value set to only a certain type.

Number Type

Number types essentially represent any real number (which includes positive, negative, and decimal numbers). They can be plainly typed in as input to any function. For example, the operation (10 + 10 + (15 - 5)) can be performed as follows:

```
add(10, 10, subtract(15, 5))
```

String Type

A string type is very much like a string in most programming languages. However, it can only be declared within double quotes. For example, consider this function which returns the length of the input string:

```
strlen("Hello World")
```

The function would appropriately return the value 11.

Boolean Type

A boolean type is simple. It can be one of two values: true or false. This type mainly acts as a flag, as in whether a condition is met or not. For example, the function contains takes in as input two strings and returns true if the first string contains the second string or false otherwise.

```
and(contains("Hello World", "llo Wo"), true)
```

The function would appropriately return the value true.

Enum Type (list)

An enum type is a collection of items. The items need to all be of the same type, they can be varied. An enum can be declared using the enum function as follows:

```
enum("hello world!", false, add(10, 15))
```

Alternatively, the createList function (an alias to enum) can be used to create enums in a formula.

```
createList("hello world!", false, add(10, 15))
```

This function would appropriately return an enum type containing "hello world!",

false, and 25 as its elements.

Link Type (relationship)

A link represents one side of a relationship and is used to access related records. For example, the accounts link field of Contacts is used to access the account_type

```
related($accounts, "account_type")
```

For most of the out-of-the-box relationships, links are named with the name of the related module, in lower case.

Follow the steps listed below to find the name of the link fields for relationships that do not follow the convention above:

1. Open the vardef file for the module you are working on: ./cache/modules/{module}/{module}vardefs.php
2. Find the link field that matches the relationship you are looking for.

Functions

Functions are methods to be used in formulas. Each function has a function name , a parameter count, a parameter type requirement, and returns a value. Some functions such as add can take any number of parameters. For example: add(1), add(1, 2), add(1, 2, 3) are all valid formulas. Functions are designed to produce the same result whether executed on the server or client.

Triggers

A Trigger is an object that listens for changes in field values and after a change is performed, triggers the associated Actions in a Dependency.

Actions

Actions are functions which modify a target in some way. Most Actions require at least two parameters: a target and a formula. For example, a style action will change the style of a field based on a passed in string formula. A value action will update a value of a field by evaluating a passed in formula.

Dependencies

A Dependency describes a set of rules based on a trigger and a set of actions. Examples include a field whose properties must be updated dynamically or a panel which must be hidden when a drop down value is not selected. When a Dependency is triggered it will appropriately carry out the action it is designed to. A basic Dependency is when a field's value is dependent on the result of evaluating a Expression. For example, consider a page with five fields with It's "a", "b", "c", "d", and "sum". A generic Dependency can be created between "sum" and the other four fields by using an Expression that links them together, in this case an add Expression. So we can define the Expression in this manner: 'add($a, $b, $c, $d)' where each field id is prefixed with a dollar ($) sign so that the value of the field is dynamically replaced at the time of the execution of the Expression.

An example of a more customized Dependency is when the field's style must be somehow updated to a certain value. For example, the DIV with id "temp" must be colored blue. In this case we need to change the background-color property of "temp". So we define a StyleAction in this case and pass it the field id and the style change that needs to be performed and when the StyleAction is triggered, it will change the style of the object as we have specified.

Sugar Logic Based Features

Calculated Fields

Fields with calculated values can now be created from within Studio and Module Builder. The values are calculated based on Sugar Logic formulas. These formulas are used to create a new dependency that are executed on the client side in edit views and the server side on save. The formulas are saved in the varies or vardef extensions and can be created and edited directly. For example, the metadata for a simple calculated commission field in opportunities might look like:

```
'commission_c' => array(
    'name' => 'commission_c',
    'type' => 'currency',
    'calculated' => true,
    'formula' => 'multiply($amount, 0.1)',
    //enforced causes the field to appear read-only on the layout
    'enforced' => true
),
```

Dependent fields

A dependent field will only appear on a layout when the associated formula evaluates to the Boolean value true. Currently these cannot be created through Studio and must be enabled manually with a custom vardef or vardef extension. The "dependency" property contains the expression that defines when this field should be visible. An example field that only appears when an account has an annual revenue greater than one million.

```
'dep_field'=> array(
    'name' => 'dep_field',
    'type' => 'varchar',
    'dependency' => 'greaterThan($annual_revenue, 1000000)',
),
```

Dependent dropdowns

Dependent dropdowns are dropdowns for which options change when the selected value in a trigger dropdown changes. The metadata is defined in the vardefs and contains two major components, a "trigger" id which is the name of the trigger dropdown field and a 'visibility grid' that defines the set of options available for each key in the trigger dropdown. For example, you could define a sub-industry field in accounts whose available values depend on the industry field.

```
'sub_industry_c' => array(
    'name' => 'sub_industry_c',
    'type' => 'enum',
    'options' => 'sub_industry_dom',
    'visibility_grid'=> array(
        'trigger' => 'industry',
        'values' => array(
            'Education' => array(
                'primary',
                'secondary',
                'college'
            ),
            'Engineering' => array(
                'mechanical',
                'electrical',
                'software'
            ),
        ),
    ),
```

),

Last Modified: 09/26/2015 04:23pm

Dependencies

Overview

The complete list of Actions available

Dependency Actions

Class	Action	Parameters	Comments
AssignToUserAction	AssignTo	value: String username.	
PanelVisibilityAction	SetPanelVisibility	target: id of panel to hide. value: Formula used to determine if the panel should be visible.	Doesn't work on tabbed layouts.
ReadOnlyAction	ReadOnly	target: Name of field to make read only value: Formula used to determine if the field should be editable.	Used by calculated fields to prevent editing.
SetOptionsAction	SetOptions	target: Name of field to make modify keys: List of option keys for the dropdown. label: List of option labels for the dropdown.	Used by Dependent Dropdowns
SetRequiredAction	SetRequired	target: Name of	

		field to make required label: id of label element for this field value: Formula used to determine if the field should be required.	
SetValueAction	SetValue	target: Name of field to modify value: Formula to get the value for target field	Used by Calculated fields
VisibilityAction	SetVisibility	target: Name of field to hide value: Formula used to determine if the field should be visible.	Used by Dynamic fields

Last Modified: 01/08/2017 05:40pm

Extending Sugar Logic

The most important feature of Sugar Logic is that it is simply and easily extendable. Both custom formula functions and custom actions can be added in an upgrade safe manner to allow almost any custom logic to be added to Sugar.

Writing a Custom Formula Function

Custom functions will be stored in ./custom/include/Expressions/Expression/{Type}/{Function_Name}.php. The first step in writing a custom function is to decide what category the function falls under. Take for example a function for calculating the factorial of a number. In this case we will be returning a number so we will create a file in ./custom/include/Expressions/Expression/Numeric/ called "FactorialExpression.php". In the new PHP file we just created, we will define a class called FactorialExpression that will extend NumericExpression. All formula functions must follow the format "{functionName}Expression.php" and the class name must match the file name. Next we need to decide what parameters the function will accept. In this case, we need take in a single parameter, the number

to return the factorial of. Since this class will be a sub-class of NumericExpression, it by default accepts only numeric types, we need not worry about specifying the type requirement.

Next, we must define the logic behind evaluating this expression. So we must override the abstract evaluate() function. The parameters can be accessed by calling an internal function getParameters() which returns the parameters passed in to this object. So with all this information we can go ahead and write the code for the function.

```
<?php

require_once('include/Expressions/Expression/Numeric/NumericExpression
.php');
    class FactorialExpression extends NumericExpression
    {
        function evaluate()
        {
            $params = $this->getParameters();
            // params is an Expression object, so evaluate it
            // to get its numerical value
            $number = $params->evaluate();
            // exception handling
            if ( ! is_int( $number ) )
            {
                throw new Exception("factorial: Only accepts
integers");
            }
            if ( $number < 0 )
            {
                throw new Exception("factorial: The number must be
positive");
            }
            // special case 0! = 1
            if ( $number == 0 ) return 1;
            // calculate the factorial
            $factorial = 1;
            for ( $i = 2 ; $i <= $number ; $i ++ )
            {
                $factorial = $factorial * $i;
            }
            return $factorial;
        }
        // Define the javascript version of the function
```

```
        static function getJSEvaluate()
        {
            return <<<EOQ
                var params = this.getParameters();
                var number = params.evaluate();
                // reg-exp integer test
                if ( ! /^\d*$/.test(number) )
                throw "factorial: Only accepts integers";
                if ( number < 0 )
                throw "factorial: The number must be positive";
                // special case, 0! = 1
                if ( number == 0 )
                return 1;
                // compute factorial
                var factorial = 1;
                for ( var i = 2 ; i <= number ; i ++ )
                factorial = factorial * i;
                return factorial;EOQ;
        }
        function getParameterCount()
        {
            return 1; // we only accept a single parameter
        }
        static function getOperationName()
        {
            return "factorial"; // our function can be called by
'factorial'
        }
    }
?>
```

One of the key features of Sugar Logic is that the functions should be defined in both php and javascript, and have the same functionality under both circumstances. As you can see above, the getJSEvaluate() method should return the JavaScript equivalent of your evaluate() method. The JavaScript code is compiled and assembled for you after you run the "Rebuild Sugar Logic Functions" script through the admin panel.

Writing a Custom Action

Using custom actions, you can easily create reusable custom logic or integrations that can include user-editable logic using the Sugar Formula Engine. Custom actions will be stored in "custom/include/Expressions/Actions/{ActionName}.php".

Actions files must end in "Action.php" and the class defined in the file must match the file name and extend the "AbstractAction" class. The basic functions that must be defined are "fire", "getDefinition", "getActionName", "getJavascriptClass", and "getJavscriptFire". Unlike functions, there is no requirement that an action works exactly the same both server and client side as this is not always sensible or feasible.

A simple action could be a "WarningAction" that shows an alert warning the user that something may be wrong, and logs a message to the sugarcrm.log file if triggered on the server side. It will take in a message as a formula so that the message can be customized at run time. We would do this by creating a php file in ./custom/include/Expressions/Actions/WarningAction.php as shown below:

```
<?php
    require_once("include/Expressions/Actions/AbstractAction.php");
    class WarningAction extends AbstractAction
    {
        protected $messageExp = "";
        function SetZipCodeAction($params)
        {
            $this->messageExp = $params['message'];
        }
        /**
        * Returns the javascript class equivalent to this php class
        * @return string javascript.
        */
        static function getJavascriptClass()
        {
            return <<<EOQ
                SUGAR.forms.WarningAction = function(message)
                {
                    this.messageExp = message;
                };
                //Use the sugar extend function to extend
AbstractAction
                SUGAR.util.extend(SUGAR.forms.WarningAction,
SUGAR.forms.AbstractAction,
                {
                    //javascript exection code
                    exec : function()
                    {
                        //assume the message is a formula
                        var msg =
SUGAR.forms.evalVariableExpression(this.messageExp);
```

```
                    alert(msg.evaluate());
                }
            });EOQ;
        }
        /**
        * Returns the javascript code to generate this actions
equivalent.
        * @return string javascript.
        */
        function getJavascriptFire()
        {
            return "new
SUGAR.forms.WarningAction('{$this->messageExp}')";
        }
        /**
        * Applies the Action to the target.
        * @param SugarBean $target
        */
        function fire(&$target)
        {
            //Parse the message formula and log it to fatal.
            $expr = Parser::replaceVariables($this->messageExp,
$target);
            $result = Parser::evaluate($expr)->evaluate();
            $GLOBALS['log']->warn($result);
        }
        /**
        * Returns the definition of this action in array format.
        */
        function getDefinition()
        {
            return array("message" => $this->messageExp);
        }
        /**
        * Returns the short name used when defining dependencies that
use this action.
        */
        static function getActionName()
        {
            return "Warn";
        }
    }
?>
```

Last Modified: 09/26/2015 04:23pm

Using Sugar Logic Directly

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 03:22am

Accessing an External API with a Sugar Logic Action

Let us say we were building a new Action called "SetZipCodeAction" that uses the yahoo geocode API to get the zip code for a given street + city + state address.

Since the Yahoo Geocode API requires JSON requests and returns XML data, we will have to write both php and javascript code to make and interpret the requests. Because accessing external APIs in a browser is considered cross site scripting, a local proxy will have to be used. We will also allow the street, city, state parameters to be passed in as formulas so the action could be used in more complex Dependencies.

First, we should add a new action that acts as the proxy for retrieving data from the Yahoo API. The easiest place to add that would be a custom action in the "Home" module. The file that will act as the proxy will be "custom/modules/Home/geocode.php". It will take in the parameters via a REST call, make the call to the Yahoo API, and return the result in JSON format.

geocode.php contents:

```
<?php
    function getZipCode($street, $city, $state)
    {
        $appID =
"6ruuUKjV34Fydi4TE.ca.I02rWh.9LTMPqQnSQo4QsCnjF5wIvyYRSXPIzqlDbI.jfE-"
;
        $street = urlencode($street);
        $city = urlencode($city);
        $state = urlencode($state);
        $base_url =
"http://local.yahooapis.com/MapsService/V1/geocode?";
        $params =
```

```
"appid={$appID}&street={$street}&city={$city}&state={$state}";
        //use file_get_contents to easily make the request
        $response = file_get_contents($base_url . $params);
        //The PHP XML parser is going to be overkill in this case, so
just pull the zipcode with a regex.
        preg_match('/\<Zip\>([\d-]*)\<\/Zip\>/', $response, $matches);
        return $matches[1];
    }
    if (!empty($_REQUEST['execute']))
    {
        if (empty($_REQUEST['street']) || empty($_REQUEST['city']) ||
empty($_REQUEST['state']))
        {
            echo("Bad Request");
        }
        else
        {
            echo json_encode(array('zip' =>
getZipCode($_REQUEST['street'], $_REQUEST['city'],
$_REQUEST['state'])));
        }
    }
?>
```

Next we will need to map the geocode action to the geocode.php file. This is done by adding an action map to the Home Module. We need to create the file ./custom/modules/Home/action_file_map.php and add the following line of code:

```
<?php
    $action_file_map['geocode'] = 'custom/modules/Home/geocode.php';
```

We are now ready to write our Action. Start by creating the file ./custom/include/Expressions/Actions/SetZipCodeAction.php. This file will use the proxy function directly from the php side and make an asynchronous call on the javascript side to the proxy.

SetZipCodeAction.php contents:

```
<?php
    require_once("include/Expressions/Actions/AbstractAction.php");
    class SetZipCodeAction extends AbstractAction{
        protected $target = "";
        protected $streetExp = "";
        protected $cityExp = "";
```

```
        protected $stateExp = "";
        function SetZipCodeAction($params)
        {
            $this->target = empty($params['target']) ? " " :
$params['target'];
            $this->streetExp = empty($params['street']) ? " " :
$params['street'];
            $this->cityExp = empty($params['city']) ? " " :
$params['city'];
            $this->stateExp = empty($params['state']) ? " " :
$params['state'];
        }
        static function getJavascriptClass()
        {
            return <<<EOQ
                SUGAR.forms.SetZipCodeAction = function(target,
streetExp, cityExp, stateExp){
                    this.street = streetExp;
                    this.city = cityExp;
                    this.state = stateExp;
                    this.target = target;
                };
                SUGAR.util.extend(SUGAR.forms.SetZipCodeAction,
SUGAR.forms.AbstractAction, {

targetUrl:'index.php?module=Home&action=geocode&to_pdf=1&execute=1&',
                exec : function()
                    {
                        var street =
SUGAR.forms.evalVariableExpression(this.street).evaluate();
                        var city =
SUGAR.forms.evalVariableExpression(this.city).evaluate();
                        var state =
SUGAR.forms.evalVariableExpression(this.state).evaluate();
                        var params = SUGAR.util.paramsToUrl({
                            street: encodeURIComponent(street),
                            city: encodeURIComponent(city),
                            state: encodeURIComponent(state)
                        });
                        YAHOO.util.Connect.asyncRequest('GET',
this.targetUrl + params, {
                            success:function(o){
                                var resp =
YAHOO.lang.JSON.parse(o.responseText);
```

```
SUGAR.forms.AssignmentHandler.assign(this.target, resp.zip);
                            },
                            scope:this
                        });
                    }
                });EOQ;
        }
        function getJavascriptFire()
        {
            return "new
SUGAR.forms.SetZipCodeAction('{$this->target}', '{$this->streetExp}',
'{$this->cityExp}', '{$this->stateExp}')";
        }
        function fire(&$bean)
        {
            require_once("custom/modules/Home/geocode.php");
            $vars = array('street' => 'streetExp', 'city' =>
'cityExp', 'state' => 'stateExp');
            foreach($vars as $var => $exp)
            {
                $toEval = Parser::replaceVariables($this->$exp,
$bean);
                $$var = Parser::evaluate($toEval)->evaluate();
            }
            $target = $this->target;
            $bean->$target = getZipCode($street, $city, $state);
        }
        function getDefinition()
        {
            return array(
                "action" => $this->getActionName(),
                "target" => $this->target,
            );
        }
        static function getActionName()
        {
            return "SetZipCode";
        }
    }
?>
```

Once you have the action written, you need to call it somewhere in the code. Currently this must be done as shown above using custom views, logic hooks, or

custom modules. This will change in the future and creating custom dependencies and taking advantage of custom actions should become a simpler process requiring little to no custom code.

Last Modified: 09/26/2015 04:23pm

Creating a Custom Dependency for a View

Dependencies can also be created and executed outside of the built in features. For example, if you wanted to have the description field of the Calls module become required when the subject contains a specific value, you could extend the calls edit view to include that dependency.

./custom/modules/Calls/views/view.edit.php

```
<?php
    require_once('include/MVC/View/views/view.edit.php');
    require_once("include/Expressions/Dependency.php");
    require_once("include/Expressions/Trigger.php");
    require_once("include/Expressions/Expression/Parser/Parser.php");
    require_once("include/Expressions/Actions/ActionFactory.php");
    class CallsViewEdit extends ViewEdit
    {
        function CallsViewEdit()
        {
            parent::ViewEdit();
        }
        function display()
        {
            parent::display();
            $dep = new Dependency("description_required_dep");
            $triggerExp = 'contains($name, "important")';
            //will be array('name')
            $triggerFields =
Parser::getFieldsFromExpression($triggerExp);
            $dep->setTrigger(new Trigger($triggerExp,
$triggerFields));
            //Set the description field to be required if "important"
is in the call subject
            $dep->addAction(ActionFactory::getNewAction('SetRequired',
array(
                'target' => 'description',
                'label' => 'description_label',
```

```
                'value' => 'true')
            ));
            //Set the description field to NOT be required if
"important" is NOT in the call subject

$dep->addFalseAction(ActionFactory::getNewAction('SetRequired', array(
                'target' => 'description',
                'label' => 'description_label',
                'value' => 'false')
            ));
            //Evaluate the trigger immediately when the page loads
            $dep->setFireOnLoad(true);
            $javascript = $dep->getJavascript();
            echo <<<EOQ
                <script type=text/javascript>

SUGAR.forms.AssignmentHandler.registerView('EditView');
                    {$javascript}
                </script>EOQ;
        }
    }
?>
```

The above code creates a new Dependency object with a trigger based on the 'name' (Subject) field in of the Calls module. It then adds two actions. The first will set the description field to be required when the trigger formula evaluates to true (when the subject contains "important"). The second will fire when the trigger is false and removes the required property on the description field. Finally, the javascript version of the Dependency is generated and echoed onto the page.

Last Modified: 09/26/2015 04:23pm

Creating a custom dependency using metadata

The files should be located in ./ custom/Extension/modules/{module}/Ext/Dependencies/{dependency name}.php and be rebuilt with a quick repair after modification.

Dependencies can have the following properties:

- hooks : Defines when the dependency should be evaluated. Possible values are edit (on edit/quickCreate views), view (Detail/Wireless views), save

(during a save action), and all (any time the record is accessed/saved).
- trigger : A boolean formula that defines if the actions should be fired. (optional, defaults to 'true')
- triggerFields : An array of field names that when when modified should trigger re-evaluation of this dependency on edit views. (optional, defaults to the set of fields found in the trigger formula)
- onload : If true, the dependency will be fired when an edit view loads (optional, defaults to true)
- actions : An array of action definitions to fire when the trigger formula is true.
- notActions : An array of action definitions to fire when the trigger formula is false. (optional)

The actions are defined as an array with the name of the action and a set of parameters to pass to that action. Each action takes different parameters, so you will have to check each actions class constructor to check what parameters it expects.

The following example dependency will set the resolution field of cases to be required when the status is Closed:

```
<?php
    $dependencies['Cases']['required_resolution_dep'] = array(
        'hooks' => array("edit"),
        //Optional, the trigger for the dependency. Defaults to
'true'.
        'trigger' => 'true',
        'triggerFields' => array('status'),
        'onload' => true,
        //Actions is a list of actions to fire when the trigger is
true
        'actions' => array(
            array(
                'name' => 'SetRequired',
                //The parameters passed in will depend on the action
type set in 'name'
                'params' => array(
                    'target' => 'resolution',
                    //id of the label to add the required symbol to
                    'label' => 'resolution_label',
                    //Set required if the status is closed
                    'value' => 'equal($status, "Closed")'
                )
            ),
        ),
```

```
        //Actions fire if the trigger is false. Optional.
        'notActions' => array(),
    );
```

Last Modified: 09/26/2015 04:23pm

Using dependencies in Logic Hooks

Dependencies can not only be executed on the server side, but can be useful entirely on the server. For example, you could have a dependency that sets a rating based on a formula defined in a language file.

```
<?php
    require_once("include/Expressions/Dependency.php");
    require_once("include/Expressions/Trigger.php");
    require_once("include/Expressions/Expression/Parser/Parser.php");
    require_once("include/Expressions/Actions/ActionFactory.php");
    class Update_Account_Hook
    {
        function updateAccount($bean, $event, $args)
        {
            $formula = translate('RATING_FORMULA', 'Accounts');
            $triggerFields =
Parser::getFieldsFromExpression($formula);
            $dep = new Dependency('updateRating');
            $dep->setTrigger(new Trigger('true', $triggerFields));
            $dep->addAction(ActionFactory::getNewAction('SetValue',
array(
                'target' => 'rating',
                'value' => $formula
            )));
            $dep->fire($bean);
        }
    }
```

Last Modified: 09/26/2015 04:23pm

Updating the Sugar Logic Cache

The updatecache.php script in the main directory traverses the Expression directory for every file that ends with "Expression.php" (ignoring a small list of

excepted files) and constructs both a PHP and a JavaScript functions cache file which resides in ./cache/Expressions/functions_cache.js and ./cache/Expressions/functionmap.php. So after you create your custom functions, you should run this script to integrate it into the entire framework. This can be done by navigating to

```
Admin > Repair > Rebuild Sugar Logic Functions
```

Last Modified: 09/26/2015 04:23pm

Extension Framework

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 04:22am

Introduction

Ext Framework

The purpose of the extension framework is to provide the capability to modify Sugar metadata, such as vardefs and layouts, in a safe way that supports installing, uninstalling, disabling, and enabling without interfering with other customizations.

How it works

Each extension point has its own directory under: ./custom/Extension/application/Ext/ and ./custom/Extension/modules//Ext. Modules can deploy files there, then these files are aggregated and compiled into a single file with a predefined name (like vardef.ext.php). An example of this is vardefs. Vardef files located in ./custom/Extension/modules/Accounts/Ext/Vardefs/ are merged into ./custom/modules/Accounts/Ext/Vardefs/vardefs.ext.php. This file will then be loaded by the part of SugarCRM that updates vardefs. The extension framework is rebuilt any time a module is installed, uninstalled, enabled, disabled, and when a Quick Repair & Rebuild is run. The file ./ModuleInstall/extensions.php

contains all of the extension mappings.

Extensions Properties

- Name : Internal name of the extension. This is used in method names such as rebuild_layouts.
- Install Definition : Section name in the manifest file
- Ext Directory : The directory containing the extension files
- Ext File : The name of the file where extension files are compiled into
- Usage : Where the extension file is used

Last Modified: 11/19/2015 04:22am

Extensions

This guide provides an overview of product features and related technologies. In addition, it contains recommendations on best practices, tutorials for getting started, and troubleshooting information for common situations.

Last Modified: 11/19/2015 03:22am

ActionFileMap

Overview

Used to map actions to files. This is handy if you want to map a file as a view outside of ./custom/modules/<module>/views/view.<name>.php.

Properties

Internal Name	actionfilemap

Manifest Installdef	action_file_map
Extension Directory	./custom/Extension/modules/<module>/Ext/ActionFileMap/
Extension File	./custom/modules/<module>/Ext/ActionFileMap/action_file_map.ext.php

Example

The following example will create a new action called 'example':

./custom/Extension/modules/<module>/Ext/ActionFileMap/<file>.php

```
<?php
    $action_file_map['example'] = 'custom/example.php';
```

Next, create your action file:

./custom/example.php

```
<?php
    //Encoded as JSON for AJAX layouts
    echo '{"content":"Example View"}';
?>
```

Finally, navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions.

Last Modified: 01/08/2017 05:40pm

ActionReMap

Overview

Used to map new actions to already existing actions.

Properties

Internal Name	actionremap
Manifest Installdef	action_remap
Extension Directory	./custom/Extension/modules/<module>/Ext/ActionReMap/
Extension File	./custom/modules/<module>/Ext/ActionReMap/action_remap.ext.php

Example

The following example will map the action 'example' to 'detailview':

./custom/Extension/modules/<module>/Ext/ActionReMap/<file>.php

```
<?php
    $action_remap['example'] = 'detailview';
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions.

Last Modified: 01/08/2017 05:40pm

ActionViewMap

Overview

Used to map additional actions for a module. Previously, all actions needed to be mapped in ./custom/modules/{module}/controller.php.

Properties

Internal Name	actionviewmap
Manifest Installdef	action_view_map
Extension Directory	./custom/Extension/modules/<module>/Ext/ActionViewMap/
Extension File	./custom/modules/<module>/Ext/Action

	ViewMap/action_view_map.ext.php

Example

The following example will map a new action:

./custom/Extension/modules/<module>/Ext/ActionViewMap/<file>.php

```
<?php
    $action_view_map['example'] = 'example';
```

./custom/modules/<module>/views/view.example.php

```
<?php
    if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
    require_once('include/MVC/View/views/view.detail.php');
    class <module>ViewExample extends ViewDetail
    {
        function <module>ViewExample()
        {
            parent::ViewDetail();
        }
        function display()
        {
            echo 'Example View';
        }
    }
?>
```

Finally, navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions mapping the view.

Last Modified: 01/08/2017 05:40pm

Administration

Overview

Used to add new administrative panels to the admin section.

Properties

Internal Name	administration
Manifest Installdef	administration
Extension Directory	./custom/Extension/modules/Administration/Ext/Administration/
Extension File	./custom/modules/Administration/Ext/Administration/administration.ext.php

More information can be found in the Administration Links section.

Example

The following example will create a new admin panel:

./custom/Extension/modules/Administration/Ext/Administration/<file>.php

```
<?php

    $admin_option_defs = array();
    $admin_option_defs['Administration']['<section key>'] = array(
        //Icon name. Available icons are located in
./themes/default/images
        'Administration',

        //Link name label
        'LBL_LINK_NAME',

        //Link description label
        'LBL_LINK_DESCRIPTION',

        //Link URL
        './index.php?module=<module>&action=<action>',
    );

    $admin_group_header[] = array(
        //Section header label
        'LBL_SECTION_HEADER',

        //$other_text parameter for get_form_header()
```

```
        '',

        //$show_help parameter for get_form_header()
        false,

        //Section links
        $admin_option_defs,

        //Section description label
        'LBL_SECTION_DESCRIPTION'
    );
```

Next, we will populate the panel label values:

./custom/Extension/modules/Administration/Ext/Language/en_us.<name>.php

```
<?php

    $mod_strings['LBL_LINK_NAME'] = 'Link Name';
    $mod_strings['LBL_LINK_DESCRIPTION'] = 'Link Description';
    $mod_strings['LBL_SECTION_HEADER'] = 'Section Header';
    $mod_strings['LBL_SECTION_DESCRIPTION'] = 'Section Description';
```

Finally, navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions and the panel will appear in the Admin section.

Last Modified: 09/26/2015 04:23pm

Dependencies

Overview

Used to add dependent actions to fields and forms that can leverage more complicated logic that is not yet available in Studio.

Properties

Internal Name	dependencies
Manifest Installdef	dependencies

Extension Directory	./custom/Extension/modules/<module>/Ext/Dependencies/
Extension File	./custom/modules/<module>/Ext/Dependencies/deps.ext.php

More information about dependency actions can be found in the Sugar Logic section.

Example

The following example will create a new required field dependency:

./custom/Extension/modules/<module>/Ext/Dependencies/<file>.php

```
<?php
    $dependencies['<module>']['<unique name>'] = array(
        'hooks' => array("edit"),
        //Trigger formula for the dependency. Defaults to 'true'.
        'trigger' => 'true',
        'triggerFields' => array('<trigger field>'),
        'onload' => true,
        //Actions is a list of actions to fire when the trigger is
true
        'actions' => array(
            array(
                'name' => 'SetRequired', //Action type
                //The parameters passed in depend on the action type
                'params' => array(
                    'target' => '<field>',
                    'label'  => '<field label>', //normally
<field>_label
                    'value' => 'equal($<trigger field>, "Closed")',
//Formula
                ),
            ),
        ),
    );
```

Once a Quick Repair and Rebuild is run, the dependency will be in effect.

Last Modified: 01/08/2017 05:40pm

EntryPointRegistry

Overview

Used to map additional entrypoints.

Properties

Internal Name	entry_point_registry.ext.php
Manifest Installdef	action_view_map
Extension Directory	./custom/Extension/application/Ext/EntryPointRegistry/
Extension File	./custom/application/Ext/EntryPointRegistry/entry_point_registry.ext.php

More information can be found in the Entry Points section.

Example

The first step is to create the actual entry point. This is where all of the logic for your entry point will be located. This file can be located anywhere you choose. For my example, I will create:

./custom/customEntryPoint.php

```
<?php
    if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
    echo "Hello World!";
```

Next, we will need to create our extension in the application extensions. This will be located at:
./custom/Extension/application/Ext/EntryPointRegistry/customEntryPoint.php

```
<?php
  $entry_point_registry['customEntryPoint'] = array(
    'file' => 'custom/customEntryPoint.php',
    'auth' => true
  );
```

Finally, navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions.

Last Modified: 01/08/2017 05:40pm

Extensions

Overview

Used to allow custom extensions within the framework. Extensions added will be used along side the extensions found in ./ModuleInstaller/extensions.php

Properties

Internal Name	extensions
Manifest Installdef	extensions
Extension Directory	./custom/Extension/application/Ext/Extensions/
Extension File	./custom/application/Ext/EntryPointRegistry/extensions.ext.php

Parameters

- section : Section name in the manifest file.
- extdir : The directory containing the extension files.
- file : The name of the file where extension files are compiled into.
- module : Optional. Determines how the framework will interpret the extension.
    - <Empty> : Will enable the extension for all modules

        - Ext : ./custom/Extension/modules/<module>/Ext/<Custom Extension>/
        - Ext File : ./custom/modules/<module>/Ext/<Extension Name>.ext.php
    - <Specific Module> : Will enable the extension for the specified module such as 'Accounts'.
        - Ext Directory : ./custom/Extension/modules/<Specific Module>/Ext/<Custom Extension>/
        - Ext File : ./custom/modules/<Specific Module>/Ext/<Extension Name>.ext.php
    - Application : enables the extension for application only.
        - Ext Directory : ./custom/Extension/application/Ext/<Custom Extension>/
        - Ext File : ./custom/application/Ext/<Custom Extension>/<Extension Name>.ext.php

Example

The following example will create a new extension called 'example'. This will create an extension directory available in ./custom/Extension/application/Ext/Example/.

./custom/Extension/application/Ext/Extensions/<file>.php

```
<?php

$extensions['example'] = array(
    'section' => 'example',
    'extdir' => 'Example',
    'file' => 'example.ext.php',
    'module' => 'application' //optional paramater
);
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will rebuild the extensions and then you will be able to add your own extension files in: ./custom/Extension/application/Ext/Example/.

Last Modified: 01/08/2017 05:40pm

FileAccessControlMap

Overview

Used to restrict specific view actions from users of the system.

Properties

Internal Name	file_access
Manifest Installdef	file_access
Extension Directory	./custom/Extension/modules/<module>/Ext/FileAccessControlMap/
Extension File	./custom/modules/<module>/Ext/FileAccessControlMap/file_access_control_map.ext.php

Example

The following example will create a new restriction for the detail view:

./custom/Extension/modules/<module>/Ext/FileAccessControlMap/<file>.php

```
<?php
    $file_access_control_map['modules']['<lowercase
module>']['actions'] = array(
        'detailview',
    );
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions.

Last Modified: 01/08/2017 05:40pm

GlobalLinks

Overview

Used to manage the global links.

Properties

Internal Name	links
Manifest Installdef	linkdefs
Extension Directory	./custom/Extension/application/Ext/GlobalLinks/
Extension File	./custom/application/Ext/GlobalLinks/links.ext.php

More information can be found in the Global Control Links section.

Examples

Adding a Link

This example will demonstrate how to add a link to the global links:

./custom/Extension/application/Ext/GlobalLinks/<file>.php

```
<?php
    $global_control_links['google'] = array(
        'linkinfo' => array(
            //String Test => URL
            $app_strings['LBL_SUGARCRM'] => 'http://www.sugarcrm.com'
        )
    );
```

Next, we will define the app_string:

./custom/Extension/application/Ext/Language/<language>.<file>.php

```
<?php
    $app_strings['LBL_SUGARCRM'] = 'SugarCRM';
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions.

Removing a Link

This example will demonstrate how to remove a global link:

./custom/Extension/application/Ext/GlobalLinks/<file>.php

```
<?php
    if (isset($global_control_links['employees']))
    {
        unset($global_control_links['employees']);
    }
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions.

Last Modified: 01/08/2017 05:40pm

Include

Overview

Used to map additional modules in the system. Normally used when module builder deploys a module.

Properties

Internal Name	modules
Manifest Installdef	beans
Extension Directory	./custom/Extension/application/Ext/Include/
Extension File	./custom/application/Ext/Include/modules.ext.php

Example

This extension is normally used when deploying custom modules. The example below shows what this file will look like after a module is deployed:

./custom/Extension/application/Ext/Include/<file>.php

```
<?php
```

```
$beanList['cust_module'] = 'cust_module';
$beanFiles['cust_module'] = 'modules/cust_module/cust_module.php';
$moduleList[] = 'cust_module';
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions.

Last Modified: 01/08/2017 05:40pm

JSGroupings

Overview

Used to add additional JavaScript groupings the system.

Properties

Internal Name	jsgroupings
Manifest Installdef	jsgroups
Extension Directory	./custom/Extension/application/Ext/JSGroupings/
Extension File	./custom/application/Ext/JSGroupings/jsgroups.ext.php

Considerations

- The grouping path you specify will be created in the cache directory.
- If you wish to add a grouping that contains a file that is part of another group already, add a '.' after the <file>.js in order to make the element key unique.

Example

The example below will add a new JSGrouping:

./custom/Extension/application/Ext/JSGroupings/<file>.php

```
<?php
    //creates the file cache/include/javascript/newGrouping.js
    $js_groupings[] = $newGrouping = array(
        'custom/file1.js' => 'include/javascript/newGrouping.js',
        'custom/file2.js' => 'include/javascript/newGrouping.js',
);
```

./custom/file1.js

```
function one(){
    //logic
}
```

./custom/file2.js

```
function two(){
    //logic
}
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions and create our JSGrouping extension. Once completed, Navigate to Admin > Repair > Rebuild JS Grouping Files. This will create the grouping file shown below:

./cache/include/javascript/newGrouping.js

```
function one(){}
/* End of File custom/file1.js */
function two(){}
/* End of File custom/file2.js */
```

Last Modified: 03/09/2016 03:43pm

Language

Overview

The Language extension adds or overrides language strings. This extension is applicable to both the application and module framework. More detailed information can be found in the Language Framework section.

Application Properties

Internal Name	language
Manifest Installdef	language
Extension Directory	./custom/Extension/application/Ext/Language/
Extension File	./custom/application/Ext/Language/<language>.lang.ext.php

$app_strings Example

The following example will create a new $app_strings label:

./custom/Extension/application/Ext/Language/<language>.<name>.php

```
<?php
  $app_strings['LBL_STRING_KEY'] = 'Label Value';
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions.

Module Properties

Internal Name	language
Manifest Installdef	language
Extension Directory	./custom/Extension/modules/<module>/Ext/Language/
Extension File	./custom/modules/<module>/Ext/Language/<language>.lang.ext.php

$mod_strings Example

The following example will create a new $mod_strings label:

./custom/Extension/modules/<module>/Ext/Language/<language>.<name>.php

```
<?php
  $mod_strings['LBL_STRING_KEY'] = 'Label Value';
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions.

Last Modified: 12/29/2015 05:10pm

Layoutdefs

Overview

Used to add or override subpanel definitions.

Properties

Internal Name	layoutdefs
Manifest Installdef	layoutdefs
Extension Directory	./custom/Extension/modules/<module>/Ext/Layoutdefs/
Extension File	./custom/modules/<module>/Ext/Layoutdefs/layoutdefs.ext.php

Example

The following example will create a new subpanel for a relationship:

./custom/Extension/modules/<module>/Ext/Layoutdefs/<file>.php

```
<?php

    $layout_defs["<module>"]["subpanel_setup"]['<subpanel key>'] =
array (
        'order' => 100,
        'module' => '<related module>',
        'subpanel_name' => 'default',
        'sort_order' => 'asc',
```

```
        'sort_by' => 'id',
        'title_key' => 'LBL_SUBPANEL_TITLE',
        'get_subpanel_data' => '<subpanel key>',
        'top_buttons' => array (
            array (
              'widget_class' => 'SubPanelTopButtonQuickCreate',
            ),
            array (
              'widget_class' => 'SubPanelTopSelectButton',
              'mode' => 'MultiSelect',
            ),
        ),
    );
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions.

Please note that if you are attempting to override parts of an existing subpanel definition, you should specify the exact index rather than redefining the entire array. An example of overriding the subpanel buttons is shown below:

```
<?php

    $layout_defs["<module>"]["subpanel_setup"]["<subpanel
key>"]["top_buttons"] = array(
        array(
            'widget_class' => 'SubPanelTopButtonQuickCreate',
        ),
    );
```

Last Modified: 01/08/2017 05:40pm

LogicHooks

Overview

Used to add functionality to certain actions, such as before saving a bean.

Properties

Internal Name	logichooks
Manifest Installdef	hookdefs
Extension Directory	./custom/Extension/modules/<module>/Ext/LogicHooks/
Extension File	./custom/modules/<module>/Ext/LogicHooks/logichooks.ext.php

More information can be found in the Logic Hooks section.

Example

The following example will create a new before_save logic hook:

./custom/Extension/modules/<module>/Ext/LogicHooks/<file>.php

```
<?php

    $hook_array['before_save'][] = Array(
        1,
        'Custom Logic',
        'custom/modules/<module>/<module>_hook.php',
        'my_hook_class',
        'example_method'
    );
```

Next, create your hook class:

./custom/modules/<module>/<module>_hook.php

```
<?php

    if (!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

    class my_hook_class
    {
        function example_method($bean, $event, $arguments)
        {
            //logic
        }
    }
```

?>

Finally, navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions and the hook will be available.

Last Modified: 09/26/2015 04:23pm

Menus

Overview

Used to manage module menus.

Properties

Internal Name	menus
Manifest Installdef	menu
Extension Directory	./custom/Extension/modules/<module>/Ext/Menus/
Extension File	./custom/modules/<module>/Ext/Menus/menu.ext.php

Considerations

- Added a $module_menu=array(); to the top of an extension file will effectively clear out any of the standard menu items compiled before it. You could then replace them all with new definitions.
- If the custom extension file is in the ./custom/Extension/application/Ext/Menus/ directory, then the menu changes will affect shortcut menus in every module.

$module_menu Parameters

- URL : The URL to navigate to. You can look at <install

dir>/<ModuleName>/Menu.php to see the shortcuts for the existing modules.
- Link Title : The label of the link.
- Image Icon : Icon for the link. Only valid for certain themes. The icons are found in ./themes/default/images.
- Module : The name of the module.

Examples

Adding a Menu Link

The following example will create a new menu link for a module:

./custom/Extension/modules/<module>/Ext/Menus/<file>.php

```
<?php

    $module_menu[] = Array(
        //URL
        "index.php?module=<module>&action=<action>",

        //Label String
        $mod_strings['LNK_EXAMPLE'],

        //Image icon. Icons are found in ./themes/default/images.
        'ExampleIcon',

        //Module Name
        '<module>'
    );
```

Next, create your label:

./custom/Extension/application/Ext/Language/<language>.<name>.php

```
<?php

    $mod_strings['LNK_EXAMPLE'] = 'Example Link';
```

Finally, navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions and the menu item will appear under the module.

Restricting Menu Items

To restrict a menu item for users based on ACL, you can do the following:

./custom/Extension/modules/<module>/Ext/Menus/<file>.php

```
<?php

    //removes the link if no access to editview
    if(ACLController::checkAccess('<module>', 'edit', true))
    {
        $module_menu[] = Array(
            //URL
            "index.php?module=<module>&action=<action>",

            //Label String
            $mod_strings['LNK_EXAMPLE'],

            //Image icon. Icons are found in ./themes/default/images.
            'ExampleIcon',

            //Module Name
            '<module>'
        );
    }
```

Navigate to Admin > Repair > Quick Repair and Rebuild. This ACL check will see if the user has access to edit the module. If they do, the link will be displayed.

Last Modified: 01/08/2017 05:40pm

ScheduledTasks

Overview

Used to add custom scheduler functions.

Properties

Internal Name	schedulers
Manifest Installdef	scheduledefs

Extension Directory	./custom/Extension/modules/Schedulers/Ext/ScheduledTasks/
Extension File	./custom/modules/Schedulers/Ext/ScheduledTasks/scheduledtasks.ext.php

More information can be found in the Schedulers section.

Example

In this example, a custom function will be added:

./custom/Extension/modules/Schedulers/Ext/ScheduledTasks/<name>.php

This file will contain our new job function:

```
<?php
    array_push($job_strings, 'custom_job');
    function custom_job()
    {
        //logic here
        //return true for completed
        return true;
    }
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions and the function will now be available.

Last Modified: 01/08/2017 05:40pm

UserPage

Overview

Used to add sections to the User Management DetailView.

Properties

Internal Name	userpage

Manifest Installdef	user_page
Extension Directory	./custom/Extension/modules/Users/Ext/UserPage/
Extension File	./custom/modules/Users/Ext/UserPage/userpage.ext.php

Example

In this example, a custom table will be added to the users DetailView:

./custom/Extension/modules/Users/Ext/UserPage/<name>.php

This file will contain our new table:

```
<?php
        $HTML=<<<HTML
        <table cellpadding="0" cellspacing="0" width="100%" border="0"
class="list view">
            <tbody>
                <tr height="20">
                    <th scope="col" width="15%">
                        <slot>Header</slot>
                    </th>
                </tr>
                <tr height="20" class="oddListRowS1">
                    <td scope="row" valign="top">
                        Content
                    </td>
                </tr>
            </tbody>
        </table>HTML;
    echo $HTML;
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions and the table will now be available under Admin > User Management > {User}.

Last Modified: 01/08/2017 05:40pm

Utils

Overview

Used to add additional functions to the global utils.

Properties

Internal Name	utils
Manifest Installdef	utils
Extension Directory	./custom/Extension/application/Ext/Utils/
Extension File	./custom/application/Ext/Utils/custom_utils.ext.php

Alternatively, additional functions can also be added by creating: ./custom/include/custom_utils.php. This method of creating utils is still compatible but is not recommended from a best practices standpoint.

Example

The example below will add a new function to the global utils:

./custom/Extension/application/Ext/Utils/<file>.php

```
<?php
    function utilFunction()
    {
        //logic
    }
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions.

Last Modified: 01/08/2017 05:40pm

Vardefs

Overview

Used to add or override vardefs in the system.

Properties

Internal Name	vardefs
Manifest Installdef	vardefs
Extension Directory	./custom/Extension/modules/<module>/Ext/Vardefs/
Extension File	./custom/modules/<module>/Ext/Vardefs/vardefs.ext.php

More information on Vardefs can be found in Module Framework under the Vardefs section.

Examples

Overriding an existing Vardef

The most common use of the vardef extensions from a customization standpoint is to override the attributes of an existing vardef. To do this, you should avoid trying to redefine the entire vardef and only update the specific index you want to change.

An example vardef is shown below. Your vardef override should not look like this unless you are creating a new vardef:

```
$dictionary['<module>']['fields']['name'] => array (
    'name' => 'name',
    'vname' => 'LBL_NAME',
    'dbType' => 'varchar',
    'type' => 'name',
    'len' => '50',
    'comment' => 'Example Vardef',
    'required' => false,
    'unified_search' => true,
    'full_text_search' => array('boost' => 3),
);
```

To update this vardef to be required, you should overwrite it by doing the following:

./custom/Extension/modules/<module>/Ext/Vardefs/<file>.php

```
$dictionary['<module>']['fields']['name']['required'] = false;
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions.

Creating Custom Fields

If your goal is to manually create a custom field on an instance, you should be using the ModuleInstaller to create the field. This can be used both for an installer package and programmatically. An example of creating a field from a module loadable package can be found under Package Examples in the Creating an Installable Package that Creates New Fields section. An example of programmatically creating a field can be found in Module Vardefs under the Manually Creating Custom Fields section.

Last Modified: 03/09/2016 03:43pm

WirelessLayoutdefs

Overview

Used to add additional subpanels to the wireless views.

Properties

Internal Name	wireless_subpanels
Manifest Installdef	wireless_subpanels
Extension Directory	./custom/Extension/modules/<module>/Ext/WirelessLayoutdefs/
Extension File	./custom/modules/<module>/Ext/WirelessLayoutdefs/wireless.subpaneldefs.ext.php

Example

The following example will add a new subpanel to a select module':

./custom/Extension/modules/<module>/Ext/WirelessLayoutdefs/<file>.php

```
$layout_defs['<module>']['subpanel_setup']['<subpanel module>'] =
array(
    'order' => 10,
    'module' => '<subpanel module>',
    'get_subpanel_data' => '<subpanel name>',
    'title_key' => 'LBL_SUBPANEL_TITLE',
);
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions.

Last Modified: 01/08/2017 05:40pm

WirelessModuleRegistry

Overview

Used to add additional modules to the available modules for mobile.

Properties

Internal Name	wireless_modules
Manifest Installdef	wireless_modules
Extension Directory	./custom/Extension/application/Ext/WirelessModuleRegistry/
Extension File	./custom/application/Ext/WirelessModuleRegistry/wireless_module_registry.ext.php

Example

The example below will add a new module (cust_module) to the list of available

modules for mobile:

./custom/Extension/application/Ext/WirelessModuleRegistry/<file>.php

```
<?php
    $wireless_module_registry['cust_module'] = array(
        //enables/disables record creation
        'disable_create' => false,
);
```

Navigate to Admin > Repair > Quick Repair and Rebuild. The system will then rebuild the extensions and the module will now be available on the mobile layout.

Last Modified: 03/09/2016 03:43pm

Migration

An overview of migrating data.

Last Modified: 11/19/2015 03:22am

Migrating From On-Demand to On-Site

Overview

Occasionally system administrators will have the need to deploy versions of their On-Demand instance to an On-Site system. Reasons for this type of deployment are:

- Testing locally developed modules
- Migrating from On-Demand to On-Site. Please note that running the 6.6 release in an On-Site environment is not supported in any capacity as this release is an On-Demand only release.

Steps to Complete

For these situations, users may request a backup of their On-Demand system by

filing a support case. When a backup is requested, SugarCRM will provide the user with an FTP account where they can download three files.

- instance_name-YYYYMMDDHHmm.sql.gz (backup of database)
- instance_name-YYYYMMDDHHmm-triggers.sql.gz (backup of database triggers)
- instance_name-YYYYMMDDHHmm.files.tgz (backup of file system)

These files can be deployed to a local system as described below.

1. Extract and import the SQL data as follows:
    - Extract the SQL file via an archive utility (e.g. 7-Zip) or via command line such as:

      ```
      gunzip instance_name-YYYYMMDDHHmm.sql.gz
      ```
    - Create a database on your MySQL server via command line:

      ```
      mysqladmin -u mysql_username -p create instance_name
      ```

      Or, if already logged into MySQL, with a command such as:

      ```
      CREATE DATABASE instance_name;
      ```
    - Import the SQL data to your MySQL server via the command line:

      ```
      mysql -u mysql_username database_name -p <
      instance_name-YYYYMMDDHHmm.sql
      ```
2. Extract the tar file to your web server's web root directory (e.g. /var/www/html) with the following command:

   ```
   tar -C /var/www/html -xzf instance_name-YYYYMMDDHHmm.files.tgz
   ```
   This will create a directory named "instance_name-YYYYMMDDHHmm" in your web root directory.
3. Rename the newly created directory:

   ```
   mv /var/www/html/instance_name-YYYYMMDDHHmm
   /var/www/html/instance_name
   ```
4. For Linux-based servers, perform the following actions:
    - Change the ownership of the directory to be accessible by the Apache user and group. Please note that the user and group (e.g. apache, www-data, etc.) may differ depending on the web server configuration.

      ```
      chown -R apache:apache /var/www/html/instance_name
      ```

- Change the permissions of the directory to ensure files can be accessed by the application. The actual permission values may differ depending on server security settings.

  ```
  chmod -R 770 /var/www/html/instance_name
  ```

5. Edit the config.php file to point to your database.
   - Open the config.php file:

     ```
     vi /var/www/html/instance_name/config.php
     ```

   - Locate and update the "dbconfig" array with the information appropriate for your MySQL server as follows:

     ```
     'dbconfig' =>
      array (
        'db_host_name' => 'localhost',
        'db_host_instance' => 'SQLEXPRESS',
        'db_user_name' => 'mysql_username',
        'db_password' => 'mysql_password',
        'db_name' => 'instance_name',
        'db_type' => 'mysql',
        'db_port' => '',
        'db_manager' => 'MysqliManager',
      ),
     ```

6. Edit the config.php file and modify the following parameters:
   - 'site_url' should be the URL of the instance on your server (e.g. http://www.mysugarinstance.com)
   - 'host_name' should be the URL of the instance without the http protocol (e.g. www.mysugarinstance.com)
7. Modify the .htaccess file in the root directory of the instance and remove the following lines if they exist:

   ```
   #Added by operations to force SSL
   RewriteEngine On
   RewriteCond  %{QUERY_STRING} !^entryPoint=WebToLeadCapture
   RewriteCond  %{HTTP:X-SSL-CLUSTER} !^od2$
   RewriteRule ^(.*)$ https://%{HTTP_HOST}%{REQUEST_URI} [R=301,L]
   ```

8. If you are migrating permanently to an On-Site deployment, there are additional modifications that should be made to ensure full functionality for your instance of Sugar.

   - To restore the ability to perform ad hoc file backups, open the ./modules/Administration/Backups.php file, and at line 1 change:

     ```
     <?php
     die("Disabled on Sugar On-Demand.  Please contact
     ```

```
support@sugarcrm.com or file a support ticket at
http://support.sugarcrm.com to request a backup.");
?>
<?php
```
To:

```
<?php
```

- To restore the ability to perform upgrades, open the ./config.php file and remove the following line:

  ```
  'disable_uw_upload' => true,
  ```
- To display the full text search configuration options under Admin > Search, open the ./config.php file and remove the following line:

  ```
  'hide_full_text_engine_config' => true,
  ```
- To display the Sugar log settings under Admin > System Settings, open the ./config.php file and remove the following line:

  ```
  'logger_visible' => true,
  ```
- To bypass security checks when installing packages via Admin > Module Loader, open the ./config.php file and change the following line from:

  ```
  'packageScan' => true,
  ```
  To:

  ```
  'packageScan' => false,
  ```

You should now have a working version of your On-Demand instance accessible from your local server.

Notes:

- The local system must use MySQL. Converting the database to another system, such as SQL Server or Oracle, requires special handling. For more information regarding this type of conversion please speak with your Customer Advocate.
- On-Site system administrators must be familiar with their stack and the referenced tools (gunzip, tar, mysqladmin, mysql, etc.). It is the system administrators responsibility to diagnose and troubleshoot issues specific to the stack (permissions, environment variables, etc.).

Last Modified: 09/26/2015 04:23pm

Migrating From On-Site to On-Demand

Overview

The following article describes the process of migrating an On-Site deployment to the SugarCRM On-Demand environment.

Requirements

A few requirements must be met before an instance can be migrated to the SugarCRM On-Demand environment:

1. The On-Site deployment must be running MySQL. If you are using SQL Server or Oracle, you will need to speak with your Sugar Customer Advocate about data migration options.
2. The instance must be updated to a supported version of Sugar. To find out if your current version is supported you can check our Supported Versions. Please refer to the appropriate Administrator/Upgrade Guide for your version of SugarCRM if you need to upgrade your instance (http://support.sugarcrm.com/02_Documentation)

How can I find the version of my SugarCRM instance?

The version of your instance can be found by navigating to the About page in your SugarCRM instance from the global links menu. Once identified you can check this version against our Supported Versions.

Migration

Once the above requirements have been met you are ready to migrate.

Support Portal

Inform SugarCRM Customer Support of your intention to migrate by opening a case through the support portal (http://www.sugarcrm.com/support/portal). They will provide you with an FTP site and a set of credentials so you can transfer your

instance. SugarCRM Customer Support will expect you to provide two files. One file will be an archive (zip, tar, etc.) containing all the files and folders of your SugarCRM instance. The second file will be an export of your SQL database; it is a good idea to archive the resulting SQL export as well.

Files and Folders

You should be aware of the location of your SugarCRM instance on your On-Site server. If you do not, you can locate the path to your instance by navigating to:

```
Admin > Schedulers
```

Once there, you should see something similar to this:

To Setup Crontab
Note: In order to run Sugar Schedulers, add the following line to the crontab file: * * * * * cd <path to sugar instance>; php -f cron.php > /dev/null 2>&1

Now that you have located your SugarCRM instance, archive the entire contents of the SugarCRM root directory using the archive utility of your choice (zip, tar, WinZip, WinRar, 7zip, etc.). The SugarCRM root directory is the directory that contains the files "config.php", "cron.php", "sugarcrm.log" and the folders "custom", "cache", "modules" among others.

Database

The following describes how to export a MySQL database using the command line utility "mysqldump". If you prefer you may choose to use a tool such as phpMyAdmin to export your database. The command to export a MySQL database is:

```
mysqldump -h localhost -u [MySQL user, e.g. root] -p[database
password] [name of the database] > backup.sql
```

If you do not know the host, username, password, or database name you may refer to the "config.php" file of your SugarCRM instance. The "dbconfig" array in the "config.php" file contains all the required information. The example above showed the following "dbconfig" array:

```
'dbconfig' =>  array (
    'db_host_name' => 'localhost',
    'db_host_instance' => 'SQLEXPRESS',
```

```
    'db_user_name' => 'sugarcrm',
    'db_password' => 'MyP@ssword',
    'db_name' => 'sugarcrm',
    'db_type' => 'mysql',
),
```

Using this information we can rewrite the command:

```
mysqldump -h localhost -u sugarcrm -pMyP@ssword sugarcrm > backup.sql
```

Upload

Finally, upload the two files to the FTP site provided by the SugarCRM Customer Support team. The instance will be deployed to the SugarCRM On-Demand environment and a URL to the instance will be provided to you.

Last Modified: 01/15/2016 10:09pm