THE INSIGHT PROJECT: SERVERLESS SYNCHRONIZATION OF DATA FOR
PERSONAL INFORMATION MANAGEMENT SOFTWARE

A Thesis in
Information Sciences and Technology
by
Ian W. Remmel

© 2010 Ian W. Remmel

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2010
The thesis of Ian W. Remmel was reviewed and approved by the following:

Mary Beth Rosson  
Professor of Information Sciences and Technology  
Thesis Advisor

Carleen Maitland  
Associate Professor of Information Sciences and Technology

Rosalie Ocker  
Professor of Practice, Information Sciences and Technology

Michael McNeese  
Professor of Information Sciences and Technology  
Professor in Charge, College of Information Sciences and Technology

*Signatures are on file in the Graduate School
ABSTRACT

With end-users increasingly using multiple devices to access their personal information, new solutions are required to make their information accessible. Cloud-based solutions like Gmail and its accompanying applications make data accessible to everyone but few of these provide the complete set of information management capabilities provided by software like Microsoft Exchange. Microsoft Exchange and similar commercial systems are prohibitively complex for the average user and for small businesses unable to afford a dedicated IT person. Further, cloud-based solutions have a number of privacy, security, and ownership issues that users should consider before releasing their personal information to the world.

The Insight Project addresses these issues by demonstrating that personal information can be shared among computers without a central server (or cloud-based solution) and that select personal information can be shared with other users at whatever granularity is desired. Moreover, the Project will demonstrate that this result can be achieved without use of costly proprietary software and that it requires minimal technical expertise of end-users.

The first portion of this work examines the issues surrounding the synchronization and distribution personal information, both on the desktop and in the cloud. Following this introductory material, I detail how personal information can be synchronized and distributed without the need of servers or cloud based services. Finally, I present several steps that will need to be taken in order to prepare The Insight Project for public use.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>viii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>ix</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1. Cloud Computing Defined</td>
<td>1</td>
</tr>
<tr>
<td>1.2. Personal Information Management</td>
<td>1</td>
</tr>
<tr>
<td>1.3. The Cloud as a Desktop Replacement</td>
<td>2</td>
</tr>
<tr>
<td>1.3.1. Privacy Concerns</td>
<td>3</td>
</tr>
<tr>
<td>1.3.2. Security Implications</td>
<td>3</td>
</tr>
<tr>
<td>1.3.3. Ownership and Control</td>
<td>4</td>
</tr>
<tr>
<td>1.4. Desktop Options</td>
<td>4</td>
</tr>
<tr>
<td>1.4.1. Outlook</td>
<td>5</td>
</tr>
<tr>
<td>1.4.2. Thunderbird</td>
<td>6</td>
</tr>
<tr>
<td>1.4.3. Novell Evolution</td>
<td>6</td>
</tr>
<tr>
<td>1.4.4. Zimbra</td>
<td>6</td>
</tr>
<tr>
<td>1.4.5. Windows and Macintosh OS X</td>
<td>6</td>
</tr>
<tr>
<td>1.5. Synchronizing Personal Information in the Cloud</td>
<td>8</td>
</tr>
<tr>
<td>1.5.1. Facebook and LinkedIn</td>
<td>8</td>
</tr>
<tr>
<td>1.5.2. Funambol</td>
<td>9</td>
</tr>
<tr>
<td>1.5.3. Google: Buzz, Gmail, tasks, calendar</td>
<td>9</td>
</tr>
<tr>
<td>1.5.4. MobileMe</td>
<td>9</td>
</tr>
<tr>
<td>1.6. Project Objectives</td>
<td>10</td>
</tr>
<tr>
<td>1.6.1. Demonstrate that personal information can be synchronized</td>
<td>11</td>
</tr>
<tr>
<td>1.6.2. Demonstrate that personal information can be shared among</td>
<td>11</td>
</tr>
<tr>
<td>1.6.3. Develop a platform on which a future, production-ready system</td>
<td>11</td>
</tr>
<tr>
<td>1.7. Design Considerations</td>
<td>12</td>
</tr>
<tr>
<td>1.7.1. Third-Party Libraries</td>
<td>12</td>
</tr>
<tr>
<td>1.7.2. Standards Compliance</td>
<td>12</td>
</tr>
<tr>
<td>1.7.3. Data Portability</td>
<td>13</td>
</tr>
<tr>
<td>1.7.4. Code Portability</td>
<td>13</td>
</tr>
</tbody>
</table>
1.7.5. Data Format ................................................................. 13
1.7.6. Web Server Functionality ........................................... 13
1.8. Chapter Outline ........................................................... 14
2. The Design and Implementation of the Insight Project .................... 15
  2.1. Core Libraries .......................................................... 16
  2.2. Data Abstraction Layer .............................................. 18
      2.2.1. Design Approaches Considered and Rejected ............... 18
      2.2.2. Chosen Solution: Templates with Polymorphism .......... 21
  2.3. Synchronization ....................................................... 24
      2.3.1. Client to Client ................................................. 24
      2.3.2. User Synchronization ......................................... 32
  2.4. Executable/User Interface .......................................... 34
  2.5. Plugins ................................................................. 34
      2.5.1. POP3 ............................................................ 34
      2.5.2. SMTP .......................................................... 35
      2.5.3. RSS ............................................................ 35
      2.5.4. XML Serializer ................................................ 35
      2.5.5. Internet Message Format .................................. 35
3. Conclusion and Future Work ............................................... 37
  3.1. Summary ............................................................... 37
  3.2. Satisfaction of Project Objectives .................................. 37
  3.3. Satisfaction of Design Requirements ................................ 38
      3.3.1. Third-Party Libraries ....................................... 38
      3.3.2. Standards Compliance ....................................... 38
      3.3.3. Data Portability .............................................. 38
      3.3.4. Code Portability ............................................. 39
      3.3.5. Data Format ................................................... 39
  3.4. Future Work ........................................................... 39
      3.4.1. Integrate with Boost.Serialization ........................... 39
      3.4.2. Better application of standards ............................ 40
      3.4.3. Native User Interfaces ...................................... 40
      3.4.4. SSL Encrypted Synchronization ............................. 40
      3.4.5. Create a tracking server for YOOHOO ....................... 40
Bibliography ........................................................................ 42
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Plugin Architecture</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>YOOHOO protocol with two hosts</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>YOOHOO protocol with three hosts</td>
<td>29</td>
</tr>
<tr>
<td>4</td>
<td>YOOHOO protocol with three hosts with HOST command</td>
<td>30</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1 Comparison of Desktop PIM Clients ................................................................. 7
Table 2 Comparison of Cloud Synchronization Options .................................................. 10
ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Mary Beth Rosson, for taking me on as an advisee when she already had a large number of students. Without her help and guidance, I would not have been able to produce this thesis. Additionally, I would like to thank my committee members, Dr. Carleen Maitland and Dr. Rosalie Ocker, as well as the IST Dean’s office staff, for their flexibility in a difficult situation.

Many thanks to Ed Brouwers, Bernie Zang, and Vince Mancuso for listening to me ramble when I needed to bounce an idea off of someone. Their willingness to let me talk things out and to offer ideas helped me move in directions I would not have seen on my own.

Further thanks to Steve Sawyer, Steve Pierce, and Chris Larkin, without whose efforts I would not have had the opportunity to attend graduate school.

Finally, I want to thank my parents for their support throughout college and for all that they taught me before that. Especially, I’d like to thank for them for all the at-the-time-undesired pushes they gave me to become the person I am today.
1. Introduction

Modern end users of computers do not use a single workstation, but in fact access their digital world via a myriad of interconnected electronic devices. Unfortunately, much of the desktop software for accessing one’s information is still mired in a single-workstation paradigm. As a result, users are increasingly turning to what have become known as “cloud” based solutions. While these solutions have been available and effective for some time, a variety of issues related to privacy, security, and ownership (about which the technical elite have been concerned for years [8]) have recently come to the attention of the general population; society may be reaching a point where it is ready to reject part or all of the “cloud” based options available and return to a desktop, albeit more network oriented, computing paradigm.

1.1. Cloud Computing Defined

Technology has changed quickly enough that the vocabulary needed to describe it cannot keep up. We use words and phrases like “cluster”, “grid computing”, “federated”, “distributed”, “peer-to-peer”, and “cloud computing” while having only a vague idea about what differentiates one from another. Most of these phrases have official definitions, but several, namely “cloud” and “peer-to-peer”, have very different meanings depending on context and on who is using the word or phrase. For the purposes of this document, cloud computing will be taken to describe the use of web-based software in which the user only sees a visual interface and has no concept of how the software actually performs its tasks. Gmail and DropBox are examples of end-user services provided by the cloud computing paradigm.

1.2. Personal Information Management

Personal Information Management (PIM) is perhaps the most desired and least developed promise of the information age, save possibly the paperless office. Personal Information (PI)
consists of all information of, about, or pertaining to a person. With respect to software support for PIM, however, the tools typically focus on management of email, contacts, events, tasks, and occasionally notes and files. Increasingly, news feeds, instant messages, and status updates (e.g. Twitter or Facebook posts) are also considered to be source data for PIM activities.

1.3. The Cloud as a Desktop Replacement

The rise of cloud computing has encouraged end users to store an increasing amount of their data on the Internet. For instance, Gmail allows users to send and receive email, keep track of and share calendars, and send and receive instant messages from any web browser. With Facebook, users can send instant messages, share contact and event information, publish status updates, and communicate asynchronously with an email-like messaging system. Google Docs and a new Microsoft service called Office Online even make it possible to edit and collaborate on documents from a web-browser-based application.

The convenience of the cloud-based options comes at a price, however. In all of these cases, a user’s personal information is stored on servers owned and maintained by the service provider. For years, the technical elite have raised an alarm about the implications of cloud computing, calling attention to the privacy, ownership, and security issues surrounding the paradigm; however, these issues and arguments are largely unknown to the majority of end users. Recently though, the problematic launch of Google Buzz (in which Google’s new social networking service automatically connected users to each other without their approval) and recent changes in Facebook’s privacy policies (in which Facebook enrolled all of its users in opt-out sharing with third-party companies and permanently made previously private information public) have brought these issues to the attention of not only mainstream society, but also the Federal Government.
1.3.1. Privacy Concerns

When Google launched its social networking service called Buzz, it automatically inferred and connected users to their “friends” based on their address book data and email history. One result was that many users were automatically connected to their work supervisors; reasonable enough as that seems at first, a consequence was that many supervisors became aware of their employees’ free time activities – information that is generally not shared with management in the work place. In a more extreme case, a blogger was automatically “friended” with her abusive ex-husband, with the result that he then had access to her address and other personal information that could be to locate her [7]. The April 2010 changes to Facebook’s privacy policy changed it such that certain types of personal information could not be made private [5]. Additionally, Facebook shared previously private information with third-party marketers until one manually opted-out of sharing such information.

1.3.2. Security Implications

In some institutions, users are able to forward email sent to their institutional email addresses to alternate addresses of their choosing. In an increasing number of cases, users choose to do this by forwarding to a personal Gmail address. Reasons for this choice vary from the convenience of Gmail’s IMAP access, to simple and functional web interface, to just finding an institution’s communication software inconvenient for personal use. While Gmail may at times be more secure than institutional email (it requires a secure connection and likely employs well-paid, well-trained employees, resources that may not be available for a small company), all messages stored by Gmail are also indexed by Gmail. Because Google uses this indexed information to tailor advertisements to a user’s interest, there is concern that an unacceptable amount of privileged and potentially identifying information may be inadvertently leaked to advertisers. For this reason,
some institutions are finding that the use of Gmail is (or may be) a violation of their computer use policies.

### 1.3.3. Ownership and Control

Once one’s information is in the cloud, one’s rights to it change. Facebook’s policy on user generated content is that the creator maintains ownership of the content, but Facebook is granted “non-exclusive, transferable, sub-licensable, royalty-free, worldwide license to use any IP content that you post” [5]. The policy goes on to say that this license ends when the content is deleted; however, it indicates the content must also be deleted by everyone with whom it was shared before the license is terminated. Facebook has been reported, and admits, to using user photos as source material for its sidebar ads [4]. Google and Yahoo users have reported instances in which their email and associated accounts were terminated or disabled without notice or explanation [3][14]. In these cases, while they still technically “owned” their content, they had no way of accessing it. In general even when users own their cloud computing data in the legal sense of the term, once they move their PI to the cloud, anything can be done with it.

### 1.4. Desktop Options

With increasing concerns about the privacy, safety and reliability of cloud computing, users are taking a greater notice of their online rights or lack thereof. As their perceived online rights continue to erode and their control over their online PI – their digital selves – decreases, we may see users returning to the use of desktop, or at least decentralized and locally-controlled computing options. For example, the Diaspora project, which is aimed at creating a decentralized alternative to Facebook that enables users to store their profile information on their own local webservers, has already raised over $200,000 (their goal was $10,000), even though the developers will not begin development until the summer of 2010. The money was raised using a
service called Kickstarter, an online service to enable crowdsourcing donations for new project ideas; the fact that Diaspora made twenty times their asking amount indicates an extremely high interest in the project and, by extension, in a cloud and social networking model in which the media originator maintains control (and not just ownership) of the media.

Though it may be six months to a year before any major result is seen from Diaspora, there are a number of desktop applications that can assist users in managing their personal information, for example maintaining integrated stores of contact information, email messages, and so on. The work carried out for this thesis takes inspiration from the services, data, and concepts offered by these modern end-user applications, so that individual users can achieve their PIM goals of integration without resorting to centralized services like Gmail or other cloud computing offerings. In the following sections, I briefly describe the most popular tools, introducing enough functionality to lay the groundwork for the PIM synchronization system presented in the main body of the thesis.

1.4.1. Outlook

Outlook is a well-known Personal Information Management (PIM) client for the desktop provided by Microsoft [10]. It includes support for a range of PI types (e.g. tasks, address book data, calendar events) and does a very good job of integrating them (e.g. a task can be generated from an email message or an address book contact).

When Outlook is used in conjunction with a Microsoft Exchange server, users can seamlessly move from one computer to another and all of their Outlook data can move with them. Unfortunately, without an Exchange server, or the use of one or more of the cloud solutions discussed later, there is no practical way to fully synchronize Outlook’s data across computers or to connect with a mobile device. Though the services discussed below provide the ability to
synchronize users’ PI, the plugins required to integrate them with Outlook have tended to be either under-developed or overpriced.

1.4.2. Thunderbird

Thunderbird is often classified as PIM client simply because it is an email client and the two are often considered one and the same. That said, with the appropriate plugins, Thunderbird can provide the same level of PIM support as Outlook sans Exchange.

1.4.3. Novell Evolution

Novell Evolution (formerly Ximian Evolution) is the de facto PIM client for the GNOME desktop [11]. Its feature set is comparable to Outlook in most respects, including synchronization. An informal review of related tools suggests that development of third-party synchronization tools for Evolution has been more extensive than that for Microsoft Outlook. However even these tools are in a very primitive state and require considerable technical expertise to be of use in a personal computing context.

1.4.4. Zimbra

Zimbra is both a desktop PIM client and PI server similar to Microsoft Exchange [17]. The desktop version of Zimbra has the same general feature set as Outlook and Evolution, but improves upon their synchronization options by providing an open source version of their server for free. Unfortunately, the server must be run on a Linux or OS X system, so the benefit for Windows users is limited.

1.4.5. Windows and Macintosh OS X

Both Microsoft Windows and Apple Macintosh OS X operating systems provide multiple small clients that support PIM functionality. Depending on the version of Windows in use, an end user
can pursue PIM tasks with features from Outlook Express or Windows Live Mail, the Windows Address Book, and Windows Calendar. Mac OS X provides Mail and iCal.

Although the Windows HomeGroup, introduced with Windows Vista, may make it possible to synchronize data, any such synchronization is limited to the local network and requires that all systems involved in the synching are being operated with Microsoft software. OS X can synchronize using MobileMe, which is able to interface with Outlook; however at present MobileMe requires a yearly subscription [1].

<table>
<thead>
<tr>
<th></th>
<th>Email</th>
<th>Address Book</th>
<th>Calendar</th>
<th>Tasks</th>
<th>Notes</th>
<th>Synchronization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outlook</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>External Service</td>
</tr>
<tr>
<td>Thunderbird</td>
<td>Yes</td>
<td>Yes</td>
<td>Plugin</td>
<td>Plugin</td>
<td>No</td>
<td>External Service</td>
</tr>
<tr>
<td>Novell Evolution</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>External Service</td>
</tr>
<tr>
<td>Zimbra</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Local Server or External Service</td>
</tr>
<tr>
<td>Windows Mail/Contacts/Calendar</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Partial</td>
<td>Local Only via Windows HomeGroup</td>
</tr>
<tr>
<td>Mac Mail/iCal</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>MobileMe</td>
</tr>
</tbody>
</table>

In summary, a number of modern desktop applications and operating systems provide most aspects of PIM (see Table 1). However, very few of them offer full synchronization of PI across devices and platforms (e.g., multiple desktop machines and mobile devices), or if they do the service requires an external server or a subscription to some outside service. The primary goal of the current work was to prototype services that could extend these applications such that the PI generated in different locations and at different times can be maintained and applied in a
consistent fashion. However, before moving to this work, I will first review the approaches to PI synchronization that can be obtained as part of the cloud computing paradigm.

1.5. Synchronizing Personal Information in the Cloud

There are many options within the arena of cloud computing for synchronizing one or more types of personal information, but few options allow users to synchronize all types of personal information. Soocial is a service for synchronizing contacts across computers and the Soocial web application; the web application provides access for phones and other mobile devices [15]. Plaxo is a similar service, but requires a fee in order to sync with Outlook\(^1\) [12]. Facebook, Funambol, Google’s services, and Apple’s MobileMe provide broader PIM and PI synchronization capabilities.

1.5.1. Facebook and LinkedIn

In terms of traditional PI tasks, Facebook and LinkedIn are analogous to directory services. They allow users to share their contact information and other “profile” information. Additionally, Facebook provides limited calendar sharing abilities (users can invite others to events, but the format is not ideal for a personal calendar). Unlike Soocial and Plaxo, which rely on the account holder to keep other users’ data updated, Facebook and LinkedIn rely on individual users to keep their own data updated, thus reducing the likelihood of data stagnation.

Many Desktop PIM clients are adding or have added support for Facebook and LinkedIn. Outlook 2010 includes the Outlook Social Connector which provides a framework for pulling

\(^1\) This fee seems to be necessary to cover the licensing fees required to work with some of Outlook’s proprietary features.
directory data from social networks (so far, only LinkedIn is supported, but a Facebook plugin is in development) [10].

1.5.2. Funambol

Funambol is a full PI synchronization service. It has support for synchronizing email, calendar information, and address books. It uses an open protocol called SyncML which is an XML format for synchronizing PI. Since the protocol is open, and not specific to Funambol, official plugins exist for Outlook, OS X, and most mobile phones, plus there are unofficial plugins for Thunderbird and Evolution.

1.5.3. Google: Buzz, Gmail, tasks, calendar

The Google suite is perhaps the most complete of the PI solutions. Not only are there tools with which most desktop PIM clients can synchronize with Google, but of the group, Google provides the most comprehensive web-based access to PI. Where the majority of the cloud-based PIM tools discussed here are primarily for synchronizing data across workstations, the Google suite is also a full-fledged PIM client with which desktop clients may interact. The Google suite includes support for email, task management, shared calendars, contact management, and document editing. Additionally, Google provides support for synchronization to mobile devices (and does not limit such support to Android-based devices, though Android devices do integrate better with Google than other devices do).

1.5.4. MobileMe

MobileMe is a for-fee service provided by Apple originally developed to help synchronize the PI on one’s iPhone with the PI on one’s computer. It is now capable of keeping OS X, the iPhone, the iPad, Windows, and Outlook data synchronized. In addition to the standard PI types discussed previously, MobileMe also provides file synchronization.
In summary (see Table 2) there are many cloud-based tools for synchronizing types of PI, but few tools exist for synchronizing all types of PI. Support for contact and calendar information is common, but integration with chat and email tools is less so. Those tools that can sync most types of PI either have limited client compatibility or require a periodic fee. As these solutions are based on cloud computing techniques, they carry with them the privacy, security, and ownership issues discussed in section 1.3. A tool is needed that can operate on multiple platforms to synchronize and share personal information without involving third-parties while requiring only limited technical abilities on the user’s part.

1.6.  Project Objectives

The Insight Project was initiated as a demonstration that PIM synchronization (across different clients and computing paradigms as described above) can be accomplished without fee-based subscriptions, with minimal technical expertise, and while maintaining control over one’s personal information. This demonstration requires that The Insight Project be capable of sending and receiving email across computing contexts, and that it support the storage and manipulation of contacts and other PI. However, as a demonstration system, the high-level requirement is to show that this capability can be achieved, rather than to develop and implement the features on a scale suitable for public use.
To these ends, The Insight Project has been developed to meet the following objectives.

1.6.1. **Demonstrate that personal information can be synchronized without a dedicated server.**

End users no longer limit themselves to a single computer but most PIM clients require users to keep their data on a single workstation or upload their personal information to the cloud in order to distribute it to other machines. The Insight Project will demonstrate that PI can be shared among clients (specifically, clients of the Insight Project, not clients in general) without use of a dedicated server or service.

1.6.2. **Demonstrate that personal information can be shared among individuals without a dedicated server.**

Traditionally, users have either maintained their address books manually or relied on a corporate directory server to keep contact information up-to-date. Cloud options like Facebook and LinkedIn shift the responsibility for data integrity from the owner of the address book to the owner of the contact information. The Insight Project will include a similar model for sharing personal information, but introduce finer-grained control of who may receive information and how much detail is shared.

1.6.3. **Develop a platform on which a future, production-ready system can be built to deliver the other objectives to end users.**

The first two objectives only specify that the Project be capable of synchronizing with other instances of the Project. In order to make the Project practical for end users, it will eventually be necessary to synchronize and share information with users of other clients. As such, the client will be designed with a modular architecture in which features may be added in the future without extensive modification to the Project’s existing source code.
1.7. Design Considerations

In addition to the objectives summarized above, a number of other goals were used to guide the prototyping and implementation of The Insight Project. Many of these were motivated by the professional concerns of software engineering, for example I was aiming for a system that has a modular design, is portable across platforms, and that can be flexibly maintained and evolved in the support of future projects.

1.7.1. Third-Party Libraries

The project’s use of libraries external to The Insight Project will be limited to libraries licensed according to the Boost [1], LGPL [2], or similar license. These licenses permit use of open-source code without requiring derivative code to be itself open-source, thus deferring the decision to release the Project’s source code to a later date. If it becomes necessary to utilize a library that does not follow this licensing policy, the implementation will encapsulate the library’s function as a plugin.

1.7.2. Standards Compliance

The Insight Project will adhere to all applicable standards; whenever possible, an open, standardized protocol will be used instead of a custom or proprietary solution. In the event that a standard does not fully encompass the needs of a given application and must therefore be circumvented, care will be taken to maintain compatibility with the standard when feasible. (Note: for the purposes of this document I will mention the name of the standard itself rather than the title of the corresponding standards document unless it is necessary to cite a specific document. For instance, the document will introduce an abbreviation like SMTP instead of the official standards documents RFC 5321, 2821, or 821. This avoids the problem of document evolution that is inconsistent with industry practices; for example, RFC 821 is the official
standards document for SMTP, but in practice other documents that extend the standard play a critical role).

1.7.3. **Data Portability**

User data will be portable from one platform (hardware or software) to another. In particular, minimal effort will be needed to move a user’s profile from one machine to another.

1.7.4. **Code Portability**

The Insight Project will not rely on any platform’s operating-system-specific features. If such a feature is used, it will be encapsulated in a plugin, so that the plugin’s use, or lack thereof, does not have a negative effect on the overall functionality of the Project. If a task can be achieved on multiple operating systems but requires different code for each, then appropriate steps will be taken to make this dependence transparent.

1.7.5. **Data Format**

The user data created by The Insight Project will be readable without use of any specialized interpretation or parsing support from project.

1.7.6. **Web Server Functionality**

The Insight Project is intended to be a desktop-based but network-oriented application. In order to synchronize data between clients, the clients must be online. In the future, it may be advantageous to provide a service with which users may synchronize their clients thus negating the need to leave an otherwise unused computer running. As such, the Project should be architected with the possibility in mind that it may someday be driving a web server instead of a desktop application.
1.8. Chapter Outline

The Insight Project is a demonstration that Personal Information can be distributed to multiple devices and shared with other users without requiring a central server accessible to every user. The remainder of this thesis will explain the implementation details of the code behind the Project and how the synchronization and sharing features work. Finally, the thesis will conclude with a discussion of the objectives and design considerations, and possible future directions for the project.
2. The Design and Implementation of the Insight Project

The Insight Project, in keeping with the third design objective of developing a base for a production ready platform, is broken into three components: five static libraries that define the core functionality and internal logical flow of the project, the executable user interface, and a number of plugins (see Figure 1). The plugins implement specific protocols, data formats, and the majority of the synchronization functionality; they can be used to extend the Project’s capabilities at a later date without needing to modify any of the core libraries.

![Figure 1 Plugin Architecture](image-url)

---

### The Insight Project

<table>
<thead>
<tr>
<th>Core Libraries (Static Libraries)</th>
<th>Utility (General Utility Function)</th>
<th>Network Interface (Basic Network Access)</th>
<th>Data Abstraction Layer (Database Interaction)</th>
<th>Plugin Manager (Library Management and Interface Specification)</th>
<th>Insight (Customization of Plugin Manager Interfaces)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable</td>
<td>GTKmm (User Interface)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plugins (Dynamic Libraries)</td>
<td>RSS</td>
<td>Internet Message Format</td>
<td>POP3</td>
<td>SMTP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Host Synchronizer</td>
<td>Email Synchronizer</td>
<td>XML Serializer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Insight Project depends heavily on the Boost C++ Libraries. Many of the libraries in the Boost collection will be integral components of the next C++ standard. Many of the capabilities that Java programmers find missing in C++ are implemented by a Boost library. The Insight Project most heavily relies on Boost’s regular expression (RegEx), network (ASIO), smart pointer (Smart Pointers), function object (Function), and signals (Signals2). Boost.SmartPointers provides automatic memory management so that variables instantiated with the `new` operator do not need to be explicitly deleted. Boost.Function provides a means of creating typesafe callbacks similar to C’s function pointers, but verified at compile time to ensure that function signatures are compatible. Boost.Signals2 is similar in syntax to Boost.Function, but allows multiple callbacks to be registered with one event such that should that event occur, all of the related callbacks are executed.

The balance of this chapter is organized as follows. Section Error! Reference source not found.2.1 provides a high level description of the core libraries of classes and methods. Section 2.2 discusses the specific approaches considered and chosen for storing data and the MySQL plugin. Section 2.3 explains the synchronization interfaces and plugins, a major focus of the project. Section 2.4 discusses the executable and section 2.5 discusses the other plugins.

2.1. Core Libraries

The Network Interface and Utility libraries contain functionality that did not fit anywhere else. Network Interface encapsulates the basic, synchronous networking functionality of Boost.ASIO for doing simple network tasks (e.g. sending and receiving mail, retrieving data via HTTP, etc.).
Utility contains a function that supports base64\(^2\) encoding and decoding, a function template and set of class templates for copying values from one data type to another (e.g. \texttt{std::string} to \texttt{boost::posix_time::ptime} or \texttt{bool} to \texttt{std::string}); it also includes several functions consisting of small code snippets that were needed on a regular basis throughout the remaining code. For a more extensive explanation of these libraries, please see the NetworkInterface and Utility namespaces in the Appendix.

The PluginManager and Insight libraries support most of the program logic and control flow. The libraries were implemented as separate code bodies to simplify reuse of the PluginManager for other projects; the PluginManager library handles loading and unloading of plugins and also defines basic class interfaces, such as \texttt{Backend} and \texttt{Object}, which are extended by plugins but are not specific to software designed for communication (The \texttt{Backend} class defines a generic interface for saving and loading data and the \texttt{Object} class defines the interface for interacting with such savable data). The Insight library defines interfaces for protocols, accounts, contacts, and events. In short, all code needed by a generic software project that may use plugins, store data, and respond to events can be found in the PluginManager library. All code needed for working with communication accounts, contacts, and events can be found in the Insight library. One important exception to these general rules pertains to the components developed to support synchronization. The basic synchronization interfaces are defined in PluginManager, specialized within the Insight library, and implemented through individual plugins.

---

\(^2\) “Base64 is a generic term for any number of similar encoding scheme that encodes binary data by treating it numerically and translating it into a base 64 representation.” [16]
The plugin manager is based on a series of articles published on the Dr. [13] website.

2.2. Data Abstraction Layer

At the start of the project, the database requirements seemed basic enough that individual SQL statements could be written manually for each data type (e.g., messages, contacts, accounts, etc.); however it quickly became apparent that a programmatic approach would be required. The main requirements of the database API are to generate SQL scripts for inserting, updating, and retrieving data in a database, as well as to allow the execution of arbitrary queries. The final version of the database API meets these requirements and additionally provides automatic schema generation. Though these general requirements remained largely unaltered throughout the design and implementation process, the details of the database API evolved through several distinct stages, with significant re-design and implementation marking each one. The final implementation was generated in April of 2010. To document the software design rationale, the following subsections present options that were considered, implemented, and rejected as well as those finally accepted.

2.2.1. Design Approaches Considered and Rejected

2.2.1.1. Boost.Serialization

Boost.Serialization is a Boost library that provides a generic interface for converting C++ class instances into arbitrarily formatted files or streams. Along with the serialization interface, the library also includes importers and exporters to convert class instances to text files, XML files, and raw binary files. Because Boost.Serialization is capable of encoding a class instance with minimal modification to the class implementation, it initially seemed the ideal solution for a database abstraction layer. It soon became apparent, however, that it would take a significant amount of development time to extend Boost.Serialization to a database format. Not only would
it be necessary to define a new input and output type to encode class instances as SQL scripts, it would also be necessary to create a set of functions for interacting with the database to determine if the instance was to be inserted or updated on save and to determine what type of class to instantiate before loading the data. Moreover, many of the issues with multiply-valued variables (discussed more extensively in 2.2.1.3) would arise as well.

2.2.1.2. Integration with single database API

After Boost.Serialization was determined to be inadequate for the Project’s needs, I considered directly integrating an existing database implementation. PostgreSQL and MySQL both have robust C++ interfaces and the SQLite C interface is adequate. Additionally, SQLite runs entirely in a flat file and does not require a database server. Unfortunately, each of these options would have required me to tailor the Insight code to a specific database implementation and to a relational model. Ideally, the database API was to be an interface between object-oriented data representation in C++ and relational data representation in the database; integration with a specific database implementation would shift too much of the relational model into the C++ code.

2.2.1.3. Polymorphism without templates

The first approach to data abstraction for The Insight Project relied on C++ support of polymorphism, whereby the same method interface is implemented for different types of Objects. However, by the time this implementation was approximately eighty percent complete, including support for saving and loading class instances to and from multiple database tables, it became apparent that the reliance on polymorphism alone was creating significant complications. The implementation included a base primitive class from which approximately ten different types were derived (These types were analogous to SQL data types such as INTEGER, TEXT, BOOLEAN, etc.). The explicit definition of these data types meant that any minor interface
changes led to several hours of work updating the source code for each type. The API was finally demonstrated to be unworkable when it became necessary to store data that consisted of multiple values, as discussed below.

An email contains a list of recipients. In C++, this relationship can be represented with something as simple as an `std::list` containing `std::string`s, but in a database, such a relationship requires one table in which each row represents a message and second table in which each row represents one recipient related to a message in the first table. It became necessary to develop a `Collection` class to represent this relationship between tables. At the time, most of the database’s primary keys were sequential integers, so saving an `Object` meant assigning it a temporary primary key, saving it to the database, allowing the database to assign it a real primary key, and then retrieving that key and updating every instance of the temporary key to the permanent value. Though this was not overly complex for singly valued data, it became quite complicated for `Collection` data due to the high quantity of locations in which updates had to occur. Every `Collection`, whether it contained sorted, keyed, or sequential data, required an `std::map` to keep track of the item’s primary key. To further complicate matters, every class derived from `Object` required five functions for interacting with Insight, each of which needed to be updated each time an `Object` was changed. These functions were the class constructor, `Serialize`, `Deserialize`, `Visualize`, and `IsAltered`. Though the existence of each of these functions may seem reasonable, each was in fact almost identical, thus making it very easy to overlook updating one or more of them. Moreover, since each of these functions called an external function sequentially on each member of the class, a single function that accepted a function pointer as an additional parameter would be just as effective while significantly reducing code bloat and maintenance complexity.
2.2.2. Chosen Solution: Templates with Polymorphism

The lessons learned from the preliminary implementations of the database API (and the frustrations caused by same) resulted in a nearly complete rewrite of the database API in April 2010. The new Data Abstraction Layer (see the DAL namespace in the Appendix) took approximately three weeks to write and test (with development of other features occurring at the same time). This version contains base classes similar to the earlier designs, but instead of explicitly defining the derivative classes, the library makes use of class templates to allow the compiler to automate much of the process.

Every class whose instances can be stored in the database must be derived from DAL::Class. In order to alleviate many of the problems created by allowing the database to assign integral primary keys, Class generates a primary key at instantiation. By default, this key is simply a base64 encoded representation of the current time, but may be, and often should be, overridden in derived classes. For example, the nature of RSS services will often result in the same message being downloaded numerous times; thus primary key generation for RSS messages is based on the GUID (a field defined by the RSS standard that should be globally unique) of the message to guarantee that only one copy of the message is stored. If instead the primary key was created using the default of the time stamp, numerous copies of the same message would be stored, clearly not the intent of the user.

Whereas the earlier implementations defined ten primitive data types, the final version has a single class template, Field, with which the compiler automatically creates class definitions for specified data types. Field is derived from FieldBase which is responsible for associating a label (i.e., a column name) with a field as well as other generic functions. The derivation from FieldBase also allows references to different specializations of Field to be stored in the
same data structure. Perhaps most importantly, instead of header and source files for ten explicitly defined primitives, there are now two files for FieldBase and one for Field, thus significantly simplifying code maintenance.

Along with Field, two other types are derived from FieldBase — Lookup and Reference. A Lookup field has many of the semantics of an enum, but simplifies storage within a database. It has a set of possible values, which are specified and verified both by C++ and by the database’s referential integrity rules. On the database side, the value is stored as an integer, but in the C++ code, it can be manipulated as either an integer or the corresponding text. A Reference typically represents a non-identifying relationship between two Objects much like the database “has-a” concept (e.g. a Message has a Reference to the Account with which it was sent or received).

As in the earlier implementations, a Collection interface is used to handle multiply-valued variables. Unlike the previous versions, however, the current implementation attempts to wrap the container classes from the C++ Standard Template Library (STL) such that a developer familiar with the STL can use Collections with minimal extra effort; instead of declaring a variable (e.g., using std::map), the variable must instead be declared using the Collection interface (e.g., DAL::Collections::Map). In addition to the standard functionality of STL containers, Collections maintain an internal index to keep track of C++ variable instances and their corresponding primary keys in the database.

Instances of Class and Field are not default constructible. The instantiation of a Class requires an std::string representing the name of its most derived class while the instantiation of a Field requires a label for the Field (i.e., the name of the column in which
the Field’s value is stored). Each of a Class instance’s component Fields must be added to a map of maps of FieldBase references. The first level of this map is keyed on the group in which the Field belongs (i.e. the name of the derived class in which the Field is declared and the name of the table in which the field is stored). The second level of the map is keyed on the Field’s label. Collections are added in a similar manner to a single-level map of Collection references keyed on the name of the Collection, which corresponds to the name of the table in which the Collection’s values are stored. These two maps are what provide most of the simplicity in this version of the database API. The DAL::Transaction class is a friend of Class and contains the protected functions GetFields and GetCollections, which return references (or const references) to their respective maps. Any class that needs to work with a Class in a programmatic way (e.g. the class in the MySQL plugin for saving data) can be derived from Transaction and use these functions to iterate over every Field and Collection in the Class without needing to know the implementation details of the Class.

The Transaction class allows for another important convenience provided by the current database API. Every class derived from Class has a corresponding “Dummy” version. To instantiate a regular derivative of Class, one must use the PluginManager API to access the corresponding factory function which then heap-allocates the class (i.e. using new) and adds it to memory management. The Dummy version of the class can be stack allocated and automatically destructs when it goes out of scope; it also cannot be saved. Its purpose is to populate the Fields and Collections maps so that the MySQL Define functor can generate the appropriate CREATE TABLE scripts. In this way, all data definitions (class members, labels for class members,
database tables, and the relationship between class members and tables) are specified by declaring field variables in the class definition and by labeling those fields in the class’s constructor.

2.2.2.1. MySQL

The chosen solution provides an interface for working with an arbitrary backend, be it an SQL database or some form of flat-file system, but the interactions with any specific backend still must be implemented by a plugin. The chosen backend for The Insight Project is a local MySQL server. Connection parameters are stored in mysql.ini in the executable’s directory.

There are two official C++ libraries for interacting with a MySQL server: the MySQL-provided Connector/C++ and MySQL++. MySQL++ was chosen over Connector/C++ mostly for ease-of-use, but the choice was also beneficial in that Connector/C++ is under the GPL license while MySQL++ is LGPL.

2.3. Synchronization

In order to meet the Projects’ objectives, two separate types of synchronization must occur. The first and most difficult is that of synchronizing data among multiple clients belonging to one user (e.g., when a user adds a new RSS feed on his or her laptop, the feed’s settings should propagate to the user’s desktop computer.). The second is that of distributing personal information to other users in a controlled manner (e.g., when a user’s phone number changes, that change should be automatically distributed to specified members of the user’s address book.).

2.3.1. Client to Client

Client to client synchronization requires that the most recent version of any given Object be distributed to the other clients. More importantly, changes should be distributed synchronously to all clients that are currently reachable and asynchronously to clients that are not reachable.
2.3.1.1. Considered Methods

As for the case of the database API, several approaches that seemed attractive at first were considered and rejected. I briefly describe these here before elaborating on the current design.

2.3.1.1.1. IMAP

The IMAP protocol is designed to store email on an accessible server and make it available to multiple clients (belonging to the same user). The protocol allows the user to upload messages to the server. When a message arrives or is deleted at the server, the event propagates to all clients (either as a direct signal if the client is online or the next time the client connects if it is not). As such, the IMAP protocol provides all of the functionality required to synchronize data between clients. However, using IMAP for data synchronization would require access to an IMAP server, which, given IMAP’s complexity and storage requirements beyond those of POP3 is uncommon for home users. It would also significantly increase the storage space and network bandwidth used by the IMAP server, and it would be necessary to encode each data item as an email message for storage on the IMAP server. Most importantly, this option would fail to satisfy the project objectives as it would require a central server.

2.3.1.1.2. Email

Initially I considered the possibility of creating a specialized email account for each client and using this account to transfer data from one client to another, but this would have required use of a central server (after a fashion), again negating a key project objective. Furthermore, though this option might provide sequential, asynchronous synchronization, the delays typical in the SMTP system would be likely to introduce long delays in synchronization across synchronous clients.
2.3.1.1.3. WebDAV

WebDAV is a protocol by which HTTP may be used to store files on a WebDAV enabled web server. Other desktop software uses WebDAV as a synchronization mechanism[18]. Like the other options, however, WebDAV would require the use of a central server. Moreover, users would either need to run their own webserver or pay for a hosting service.

2.3.1.2. Chosen Method

In order to meet the objective of a serverless infrastructure, it became apparent that some form of peer-to-peer networking would be required. After examining and eliminating existing peer-to-peer technologies (e.g., BitTorrent), development began on a custom protocol. This protocol is divided into two parts: the YOOHOO Protocol for host discovery and the YOOHOO-SYNC protocol used for the actual distribution of data amongst hosts.

2.3.1.2.1. The YOOHOO Protocol

From the “original” peer-to-peer protocol Napster to current versions of BitTorrent, such protocols require a server of some kind to handle the initial network discovery process; this is because for a new client to join the network, it must have a priori knowledge of at least one entry point to the network. Similarly, the YOOHOO protocol requires that there be at least one host whose IP address or host name is known a priori. That said, the same principal host does not need to be known to all hosts.

The YOOHOO protocol consists of four commands: YOOHOO, DISCOVER, HOST, and HOSTS. These commands are used to distribute the information necessary to reestablish connections when Insight is started. When The Insight Project starts, it checks its table of known hosts and attempts to initiate sessions with each of them.
2.3.1.2.1.1. **YOOHOO**

The *YOOHOO* command is used to initiate an asynchronous session with a remote host. As schematized in Figure 2, *YOOHOO* carries with it the local host’s unique host id, routable host name or IP address, and the port number on which the local host can be reached in event of a lost connection. When a remote host receives a *YOOHOO*, it adds or updates the local host’s information to its known hosts table and to its list of active sessions. Then, it sends a *YOOHOO* back to the local host. The local host then repeats the process the remote host completed and sends a second *YOOHOO* to the remote host.

2.3.1.2.1.2. **Discover & Hosts**

Once a session has been fully created, the local host should issue a DISCOVER command to retrieve a list of the hosts of which the remote host is aware. When the remote host receives a  

---

3 This second *YOOHOO* is not strictly necessary, and should in fact be removed in future versions, but it simplifies the coding process.
DISCOVER command, it replies using a HOSTS command parameterized with its list of known hosts. When the local host receives HOSTS, it attempts to initiate sessions with each new host; the process will continue until all hosts are aware of (or fail to reach) all other hosts.

2.3.1.2.1.3. HOST

The HOST command can optionally be sent by a local host to each of its connected hosts when it receives YOOHOO. Through this protocol, a connected host can initiate sessions with the remote host, thereby reducing the number of hosts with which the remote host needs to initiate connections when it receives HOSTS. Figure 3 shows how the YOOHOO protocol functions without use of the HOST command and Figure 4 shows how it works with the HOST command. It should be noted that although use of the HOST command converges to a full set of remote hosts more quickly, it changes the nature of the connection information. For instance in Figure 4, Host A never explicitly receives Host C’s host list; assuming Hosts B and C have converged, however, the host lists on Host B and Host C will be identical, thus Host A receives Host C’s host list in an implicit fashion from Host B.
Figure 3 YOOHOO protocol with three hosts
2.3.1.2.2. YOOHOO-SYNC

YOOHOO-SYNC extends the YOOHOO protocol with two additional commands, SUBMIT and DELETE, each of which can operate in a number of modes. The protocol is a combined push/pull architecture. When all clients are online, YOOHOO-SYNC operates in a push style; when a client returns from the offline state, it operates in a pull style until its data matches that of the other online clients and then begins operating in a push style.

2.3.1.2.2.1. Online Mode

The Insight Project contains six signals to which the synchronizer reacts: object_changed, object_not_changed, object_committed, object_updated, and
object_delete_pending, and object_deleted, object_primaryKey_changed. Each signal carries with it the primary key of the object in question. When the synchronizer sees object_changed, it encodes the Object’s changed fields in an XML document and stores it as a string in a map (StagedChanges) keyed on the Object’s primary key. In case the change is sent to a client that doesn’t have a copy of the object, the object’s type is encoded as well. If object_not_changed is detected, the string is discarded from the map. When object_updated or object_committed is detected, the string is moved from StagedChanges to the CommittedChanges queue. A similar process occurs for object_delete_pending and object_deleted save that instead of encoding the object, the date and primary key are encoded. While CommittedChanges is not empty, it works through CommittedChanges sequentially to submit the changes to the other clients.

When a change is received from another client, the synchronizer first attempts to load the specified primary key from the database. Failing that, it creates a new object and takes the values from the XML received from the remote client and passes them to the new object. If the change is a delete, the specified object is deleted.

2.3.1.2.2.2. Offline Mode

The above protocol works when all clients are reachable at all times. This, of course, is not the case. When a client comes online for the first time, it needs to be given the IP address of a host with which to synchronize. Each time one client connects to another, they exchange lists of primary keys and last modified times. Once each client knows what the other client is missing, it sends full XML-encoded copies of those messages to the other client.

Once a YOOHOO session is initialized, the local host REQUESTs DELETES from the remote host; the request contains the primary key and deletion time of every Object stored on the local host.
The remote host responds by submitting **DELETES**, which likewise contains the primary key and deletion time of each **Object** on the remote host. Each host then deletes the **Objects** that have been deleted by the other host. Then, the local host **REQUESTs** **CHANGES** and the remote host responds by submitting **CHANGES**. Similarly to **DELETES**, **CHANGES** includes the primary key and last modified time of every **Object** on the respective host. Once the local host determines which host has the newer version of an **Object**, it either **REQUESTs** the **CHANGE** for each outdated primary key or **SUBMITs** the **CHANGE** for the more recent primary key. Once this exchange has completed, the session is left open and each time an **Object** is modified or deleted on either host, the host **SUBMITs** the **CHANGE** or **DELETE** to the other host (**SUBMIT CHANGE** is used to submit a new **Object** as well a changed **Object**). In the event a primary key changes (for example, when a Person’s email address is updated), the **KEYCHANGE** is **SUBMITed** to the remote host before the **CHANGE** is submitted. The primary key in the **CHANGE** however, will still match the old key. As such, each host must maintain a mapping of old keys to new keys. Once a host receives a **CHANGE** for an **Object** that has a changed primary key, it opens the **Object**, applies the changes, and saves the **Object**. Then, it alters the primary key and saves the **Object** again.

### 2.3.2. User Synchronization

Like the client synchronization method, data is synchronized between users whenever its values are changed. The difference is that instead of using a direct connection to distribute the change, the corresponding changes are broadcast via email. In this case, email has several advantages over direct connection. By using email, the receiving user does not need to be a user of The Insight Project; email is a plain-text format, so a non-Insight user can read an incoming change set and choose to apply the updates manually. Also, email is asynchronous. By depositing a
change set on the recipient’s mail server, updates can be retrieved whenever an Insight user who
is the recipient next runs Insight. For privacy reasons, recipients of change sets are blind carbon
抄送 so that one recipient cannot see which other recipients received the change set.

Also unlike machine synchronization, in which nearly all data is synchronized to all clients, only
a subset of PI is broadcast to other users; the sending user directly specifies with which other
users he or she wishes to synchronize. At present, the user-sharable PI includes only the users’
events and contact updates. For events, there are three levels of granularity: NONE, FREE/BUSY, and ALL. NONE, of course, does not broadcast any event data. FREE/BUSY broadcasts the time and date of each event, but not the title or body. ALL broadcasts time, date, title and body. For contact information, the principal user may enable or disable sharing for each available field.

Once a change set has been processed, the machine synchronizer will also broadcast it to
connected clients. These other clients may (or may not) have seen the change set. Normally,
when an Object is updated in the database, the current time and date is set as its modification
time. However, when processing change sets, the modification date used is the date of the change
set. With this date, the connected clients can determine whether or not they have already
processed the change set and react accordingly.

---

4 If a change set is received via email by client A while client B is offline, when B comes online, it will receive the change set via email when it checks mail and it will receive it from client A when it asks for updates.
2.3.2.1. XML vs. vCard/iCal

Synchronization messages are identified by a custom header field named “X-INSIGHT-CONTROL-MESSAGE”. When email messages with this field are received, they are passed to a second parser for specialized processing. For ideal interoperability, these messages should simply be vCard or iCal formatted representations of the changed Contact or Event, as these are standard formats used by many existing PIM clients. Instead, data is represented in a generic XML format. This design choice sacrifices interoperability for data-richness: the vCard format puts limits on the number of phone numbers and email address that can be recorded for a user. Encoding a contact as XML bypasses such limits. The iCal format was avoided primarily due to time constraints.

2.4. Executable/User Interface

The executable is responsible for initializing the core libraries and providing user interactions. It relies on gtkmm, the C++ bindings for the GTK+ toolkit. GTK+ is the user interface toolkit used by the GNOME desktop and many open source applications. One of the principal benefits of using gtkmm over the Microsoft .NET Framework is that while .NET only works on Microsoft operating systems, interfaces written with gtkmm can be used Windows, Linux, and OS X.

2.5. Plugins

While the previous sections explain in detail the components of the Project responsible for synchronizing and sharing users’ PI, the following plugins were developed to support the generation and interaction with the PI.

2.5.1. POP3

The POP3 plugin implements all of the required commands specified in STD 53 (RFC 1939) and all of the optional commands save APOP. APOP is a method for securely authenticating to a
POP3 server, but it does not encrypt any traffic except for the user’s password. Also, it is not commonly used. Instead, most systems handle authentication securely by simply encrypting all traffic between the client and server using TLS and then by using the standard POP3 authentication mechanism.

### 2.5.2. SMTP

The Insight Project’s implementation of the SMTP protocol includes all of the commands specified in STD 10 (RFC 831) as well as AUTH PLAIN and AUTH LOGIN as specified by RFC 2554.

### 2.5.3. RSS

The Project Objectives do not mandate an RSS plugin. Instead this plugin was created because RSS is a simple protocol with a simple format. As such, it was ideal for debugging. The RSS format is a simple, singly-nested xml document with 16 fields\(^5\). Most of the parsing for RSS relies on libxml++. RSS data is transferred via the HTTP protocol. Insight’s HTTP implementation is adapted from the HTTP example for the Boost.ASIO library.

### 2.5.4. XML Serializer

The XML serializer is used by the client synchronization plugin to convert an Object to and from XML. This plugin uses libxml++.

### 2.5.5. Internet Message Format

The POP3 and SMTP plugins exist for the purpose of transferring Internet Messages. The Internet Message Format plugin implements the majority of RFC 2822. Due to limitations of the

---

\(^5\) Insight implements RSS 0.91[9]
underlying Data Abstraction Layer, in the event that a message has multiple optional header fields with the same name, only the last-parsed field will be stored. This is not, however, an egregious divergence from RFC 2822 as the optional fields tend to be useful primarily for debugging.
3. Conclusion and Future Work

3.1. Summary

The Insight Project’s Design Objectives were to synchronize data among multiple clients belonging to a given user, share that user’s public information with other users, and to develop a platform on which a future, production-ready system can be based. YOOHOO-SYNC and the User Synchronization components meet the first two objects. The third object has been partially met, but further refinement of the Data Abstraction Layer is required before a production-ready system can be developed. Specifically, the Data Abstractions Layer must more seamlessly emulate containers from the C++ Standard Template Library (e.g. a user must be able to work with a DAL::Collections::Map identically to an std::map). I turn now to a more detailed discussion of what has been accomplished in the Insight Project thus far.

3.2. Satisfaction of Project Objectives

The Insight Project’s two main objectives were to demonstrate that PI can be shared among clients belonging to an individual user and to selectively share that user’s PI with other users. The YOOHOO-SYNC protocol achieves the first objective by distributing PI between a user’s computer via direct connections and the second objective is achieved by email PI updates to other users. The third objective, that of developing a platform from which a production-ready client can be developed, can be considered partially achieved. Though the core libraries can be used to develop further software, the core libraries can best be utilized as the “version 1.0” from which “version 2.0” can subsequently develop; the same concepts can be used but some further design work should be put into simplifying the API for plugin developers.
3.3. Satisfaction of Design Requirements

3.3.1. Third-Party Libraries

Third-party libraries include the Boost C++ Libraries, OpenSSL, gtkmm, MySQL++, and libxml++2. The Boost libraries are released under the Boost Software License [2] which is similar to the LGPL [6] while the rest are LGPL licensed.

3.3.2. Standards Compliance

The Insight Project maintains compliance with most applicable standards (POP3, SMTP, RSS, HTML, etc.), but did notably disregard two, iCal and vCard, and ignored a part of InternetMessageFormat.

When examining the vCard standard, it is quite apparent that it was developed for the corporate world; many of its fields are not used by the average user.

InternetMessageFormat allows for the addition of an unlimited number of optional fields not specified in the standard. According to the standard, these fields can repeat. Due to limitations of the underlying database API (DAL does not implement an interface for std::multimap), only the last-parsed instance of an optional field is stored and all other instances are disregarded. Though this is non-compliant, it should not affect end-user experience in most cases; also, the raw message is stored in its field in the database, so that field is not lost it is simply programmatically inaccessible.

3.3.3. Data Portability

Since data is stored in a MySQL database, a MySQL dump can be used to move data from one machine to another.
3.3.4. Code Portability

The only platform specific code in The Insight Project is contained in the PluginManager::Library class. This class is responsible for low level access to dynamic libraries (i.e. plugins). Windows and Linux use their own native implementations for working with dynamic libraries. As such, preprocessor directives are used to compile the appropriate code for the environment in use.

3.3.5. Data Format

Since data is stored in a MySQL database, standard MySQL tools can be used to read data without using software from The Insight Project.

3.4. Future Work

Although The Insight Project did accomplish it’s first two objectives, several tasks remain to make it useful in a production environment.

3.4.1. Integrate with Boost.Serialization

Even after multiple designs and revisions, the database API is still more difficult to use than it should be. Specifically, the Collection interface is too complicated. Collections can only contain certain types of data and Collections of Collections are not possible. Conversely, Boost.Serialization can convert classes with arbitrarily deep STL collections into XML (e.g. a map of maps of lists can be automatically converted to a three-element-deep XML document) with minimal effort on the developer’s part. In the long run, it should be possible to create a new input and output type for Boost.Serialization that can encode classes as SQL scripts instead of XML. Then, a set of wrapper functions would be required to determine what type of object to allocate before loading the object. In the shorter term, it may be easier to write a function to transform the XML created by Boost.Serialization into an SQL script; the same wrapper functions for memory management would be required.
3.4.2. Better application of standards

As mentioned previously iCal and vCard are not currently used by The Insight Project. Rather than discarding them, in the future, the Event and Contact classes should more closely match their formats and extend them as necessary.

3.4.3. Native User Interfaces

While interfaces written in gtkmm look relatively familiar to users on Windows, OS X, or Linux, on Windows and OS X, subtle differences tend to make it apparent that the interface does not match native programs when using Windows or OS X; generally, this is limited to things like buttons simply looking like they do not belong, but in some cases, things may be in unexpected places. For example, early versions of Firefox placed options in the Edit menu on all operating systems, but now, on Windows, options is located in the Tools menu because that is the standard location for the operating system. Eventually, interfaces written using Microsoft’s .NET framework and Apple’s Cocoa should make users feel more comfortable using The Insight Project.

3.4.4. SSL Encrypted Synchronization

At present, synchronization messages passed between clients are transferred in an unencrypted TCP stream. While this is adequate for demonstration purposes, it may pose a security and privacy risks in real world networked environment.

3.4.5. Create a tracking server for YOOHOO

The YOOHOO protocol, though “serverless” in that there is no single server on the Internet to which all clients connect, does require that at least one host in a client’s list of known hosts is reachable in order to locate the other hosts and to stay synchronized. One consequence is that if every machine changes network location while offline, no machine will be able to reach any other
without user intervention. One possible solution to this problem would be to provide a server on
the Internet for the express purpose of maintaining the locations of other hosts. In this way, hosts
could still transfer data in a peer to peer fashion, but would not need to need to know the exact
address of other host \textit{a priori}. Initially, this server’s only purpose would be host location, but
could eventually be used as a remote repository and/or web interface for one’s PI.
Bibliography


Appendix: API Documentation

Namespace Documentation

DAL Namespace Reference

The DAL namespace contains the Data Abstraction Layer.

Namespaces

- namespace Collections
- Defines specializations of STLCollection.

Classes

- class Class
  A class represents a table (or collection of tables) within a database. class Collection
- The Collection class provides an interface for working with multiple values in the same variable. class Field
- class FieldBase
- class Lookup
  The Lookup class represents a blend between an enum and a DAL::Reference. It has fixed values like an enum, but stores them in a table. class Reference
- Represents a foreign key relationship in a database. class STLCollection
- Wrapper template from which other Collections are derived/specialized. Amongst other things, provides wrappers for STL iterators so that they can work with rest of DAL. class Transaction

As a friend of Class, classes needing programmatic access to a a Class’s members must be derived Transaction.

Detailed Description

The DAL namespace contains the Data Abstraction Layer.

DAL::Collections Namespace Reference

Defines specializations of STLCollection.

Classes

- class Map
  Wraps std::map<KEY_TYPE, VAL_TYPE> in a DAL::Collection. class ReferenceSequence
- Specialize DAL::Sequence<VAL_TYPE> for VAL_TYPE = DAL::Reference. class Sequence
  Wraps std::list<VAL_TYPE> in a DAL::Collection.

Detailed Description

Defines specializations of STLCollection.
Framework Namespace Reference

The **Framework** namespace classes for managing instances of the **PluginManager** interfaces.

**Namespaces**
- namespace **Interface**

  The **Interface** namespace is the future location of the interfaces defined in **PluginManager**.

- namespace **Managers**

  The **Managers** namespace contains classes for managing each of the **PluginManager** interfaces.

**Classes**
- class **Librarian**
- class **Logger**
- class **Platform**
- class **Registry**

**Detailed Description**

The **Framework** namespace classes for managing instances of the **PluginManager** interfaces.

Framework::Interface Namespace Reference

The **Interface** namespace is the future location of the interfaces defined in **PluginManager**.

**Classes**
- class **Synchronizer**

  Defines an interface for keep changes synchronized across clients.

**Detailed Description**

The **Interface** namespace is the future location of the interfaces defined in **PluginManager**.

Framework::Managers Namespace Reference

The **Managers** namespace contains classes for managing each of the **PluginManager** interfaces.
Classes

- class BackendManager
  - manages the available backends
- class ListenerManager
  - Manages the available listeners.
- class ObjectManager
  - Keeps track of allocated Objects and provides functions for retrieving and saving them.
- class SerializerManager
  - Keeps track of Serializers and provides the functions for adding, using, and removing them.

Detailed Description

The Managers namespace contains classes for managing each of the PluginManager interfaces. Most of the functions of the various managers could conceptually belong in Platform, but this made it easier to keep track of what functions do what.

Insight Namespace Reference

The insight namespace contains specializations and derivations of PluginManager interfaces for communications related tasks.

Namespaces

- namespace Communication
  - Contains the classes and methods doing communication related tasks.
- namespace Directory
  - Contains the classes and methods for contact management.
- namespace HostSynchronzer
  - The HostSynchronizer namespace space provides functions and classes that extend the Yoohoo protocol to synchronize data amongst connected clients.
- namespace InternetMessageFormat
  - Contains the functions and methods for working with RFC 2821 formatted Internet Messages.
- namespace MySQL
  - The MySQL namespace contains the functions and classes necessary for interacting with an SQL database.
- namespace POP3
  - Defines the functions and classes necessary for retrieving messages via the POP3 protocol.
namespace RSS

Defines the functions and classes necessary for retrieving RSS news Items.

namespace Schedule

Contains classes and methods for working with schedule data.

namespace SMTP

contains the functions and classes necessary for sending mail via the SMTP protocol

namespace Synchronization

Contains interfaces from which to derive classes for synchronizing data.

namespace XML

The XML namespace contains the classes and functions for converting Objects to and from XML.

Classes

- class Frontend
  - Adds signals to PluginManager::Frontend. class Platform
  
  This class is provides the programmatic interface by which User Interfaces should interact with Insight.

Detailed Description

The insight namespace contains specializations and derivations of PluginManager interfaces for communications related tasks.

Insight::Communication Namespace Reference

Contains the classes and methods doing communication related tasks.

Classes

- class Account
  - Defines an interface from which communication accounts (e.g. POP3, SMTP, etc) can be derived. class DummyAccount

- class Message
  - Defines an interface from which communication messages (email, rss, etc) can be derived. class DummyMessage

- class Protocol
  - implements a communications path between a client and a server class Receiver

  specialization of Protocol for protocols the receive messages class Sender

  specialization of Protocol for protocols the send messages
Detailed Description
Contains the classes and methods doing communication related tasks.

Insight::Directory Namespace Reference
Contains the classes and methods for contact management.

Classes
- class **Contact**
  * Defines an Object for storing Contact information.
- class **DummyContact**
- class **Person**
  * adds content to Contact that makes sense for people but not for companies or organizations
- class **DummyPerson**

Detailed Description
Contains the classes and methods for contact management.

Insight::HostSynchronzer Namespace Reference
The HostSynchronzer namespace space provides functions and classes that extend the Yoohoo protocol to synchronise data amongst connected clients.

Detailed Description
The HostSynchronzer namespace space provides functions and classes that extend the Yoohoo protocol to synchronise data amongst connected clients.

Insight::InternetMessageFormat Namespace Reference
Contains the functions and methods for working with RFC 2821 formatted Internet Messages.

Classes
- class **Message**
  * Represents an RFC 2821 formatted Internet Message.
- class **Parser**
  * Parses raw Internet Messages into Objects.

Detailed Description
Contains the functions and methods for working with RFC 2821 formatted Internet Messages.
Insight::MySQL Namespace Reference

The MySQL namespace contains the functions and classes necessary for interacting with an SQL database.

Namespaces
- namespace Transactions

   The Transactions namespace defines functors for common SQL functions.

Classes
- class Backend
  Adapts MySQL++ to the PluginManager::Backend interface. class SQLTransaction provides a base class from which to specialize SQL transactions.

Detailed Description
The MySQL namespace contains the functions and classes necessary for interacting with an SQL database.

Insight::MySQL::Transactions Namespace Reference

The Transactions namespace defines functors for common SQL functions.

Classes
- class Define
  This is a functor that generates make table scripts from Objects.
- class Exec
- class Insert
- class Query
- class Select
- class Update
  Creates an SQL select script for the specified Object and then loads the retrieved data into the Object.

Detailed Description
The Transactions namespace defines functors for common SQL functions.

Insight::POP3 Namespace Reference
Defines the functions and classes necessary for retrieving messages via the POP3 protocol.

Classes

- class Account
  Contains the functions necessary to retrieve mail via the POP3 protocol.

Detailed Description

Defines the functions and classes necessary for retrieving messages via the POP3 protocol.
Contains the classes and methods necessary for interacting with a POP3 server.

---

Insight::RSS Namespace Reference

Defines the functions and classes necessary for retrieving RSS news Items.

Classes

- class Account
- defines an RSS Account class Item
  Defines an RSS message.

Detailed Description

Defines the functions and classes necessary for retrieving RSS news Items.

---

Insight::Schedule Namespace Reference

Contains classes and methods for working with schedule data.

Classes

- class Event
  Defines an Object for storing calendar items.

Detailed Description

Contains classes and methods for working with schedule data.

---

Insight::SMTP Namespace Reference

contains the functions and classes necessary for sending mail via the SMTP protocol

Classes

- class Account
**Detailed Description**
contains the functions and classes necessary for sending mail via the **SMTP** protocol

---

**Insight::Synchronization Namespace Reference**
Contains interfaces from which to derive classes for synchronizing data.

**Detailed Description**
Contains interfaces from which to derive classes for synchronizing data.

---

**Insight::XML Namespace Reference**
The **XML** namespace contains the classes and functions for converting Objects to and from **XML**.

**Classes**
- class **Serializer**
  *Converts Objects to and from **XML**.*

**Detailed Description**
The **XML** namespace contains the classes and functions for converting Objects to and from **XML**.

---

**Networking Namespace Reference**
The networking namespace is the future location of all networking code.

**Classes**
- class **Session**
  *The **Session** class provides a base from which to derive asynchronous protocols.*

**Detailed Description**
The networking namespace is the future location of all networking code.
NetworkInterface Namespace Reference

The NetworkInterface namespace provides a wrapper to boost.asio's synchronous networking capabilities.

Classes
- class Connection

Enumerations
- enum Security_t

Functions
- boost::shared_ptr< Connection > MakeConnection (const std::string &server, const int port, Security_t security=PLAIN)
  Wrapper function for generating a Connection object.

Detailed Description
The NetworkInterface namespace provides a wrapper to boost.asio's synchronous networking capabilities.

Enumeration Type Documentation

text

Function Documentation
boost::shared_ptr< Connection > NetworkInterface::MakeConnection (const std::string & server, const int port, Security_t security = PLAIN)

Wrapper function for generating a Connection object.

Parameters:
server The IP or hostname to which to connect
port The port on which to connect
security whether to use a plain or encrypted connection

Returns:
A shared pointer to a Connection object
Referenced by Insight::Utility::Session::Session().

Here is the caller graph for this function:

NetworkInterface::MakeConnection
Insight::Utility::Session::Session
PluginManager Namespace Reference

The PluginManager namespace contains the classes from which Plugins are defined.

Namespaces
- namespace Exception
  Contains exceptions used by the PluginManager namespace.

Classes
- class Backend
  This class handles the storage and retrieval of data. class Frontend
  This has handles external interactions with the system.
- class IniParameters
  Interacts with the INI file.
- class Library
  The Library class provides a low-level interface for working with dynamic libraries. struct LibraryData
  Contains Data about a Library. class Listener
  Generic class from which signal handling classes should be derived.
- class Frontend
  This class handles external interactions with the system.
- class IniParameters
  Interacts with the INI file.
- class Library
  The Library class provides a low-level interface for working with dynamic libraries. struct LibraryData
  Contains Data about a Library. class Listener
  Generic class from which signal handling classes should be derived.
- class Frontend
  This class handles external interactions with the system.
- class IniParameters
  Interacts with the INI file.
- class Library
  The Library class provides a low-level interface for working with dynamic libraries. struct LibraryData
  Contains Data about a Library. class Listener
  Generic class from which signal handling classes should be derived.

Used for storing version numbers. Typedefs
- typedef void(* fn_activate)(RegistrationServices *regServices, PlatformServices *services)
  Function pointer defined in plugins and used by Librarian::Load().
- typedef void(* fn_deactivate)(RegistrationServices *regServices, PlatformServices *services)
  Function pointer defined in plugins and used by Librarian::Activate().
- typedef Backend *(* fn_backend_factory)(PlatformServices &services)
  Factory signature for creating Backends.
- typedef Frontend *(* fn_frontend_factory)(PlatformServices &services)
  Factory signature for creating Frontends.
- typedef Object *(* fn_object_factory)(PlatformServices &services)
  Factory signature for creating Objects.

Enumerations
- enum Mode_t
  Enum used by serializers for specifying how much of an Object to serialize.

Functions
- PLUGIN_API LibraryData * register_plugin()
  Implemented in every plugin.
- PLUGIN_API void activate_plugin(RegistrationServices *regServices, PlatformServices *services)
Implemented in every plugin.

- PLUGIN_API void deactivate_plugin (RegistrationServices *regServices, PlatformServices *services)
  Implemented in every plugin.

---

**Detailed Description**

The **PluginManager** namespace contains the classes from which Plugins are defined.

---

**Typedef Documentation**

**typedef void(* PluginManager::fn_activate)(RegistrationServices *regServices, PlatformServices *services)**

Function pointer defined in plugins and used by Librarian::Load().

**Parameters:**

- *regServices* pointer to instance of RegistrationServices
- *services* pointer to instance of PlatformServices

**typedef Backend(*)(* PluginManager::fn_backend_factory)(PlatformServices &services)**

Factory signature for creating Backends.

**Parameters:**

- *services*

**Returns:**

**typedef void(* PluginManager::fn_deactivate)(RegistrationServices *regServices, PlatformServices *services)**

Function pointer defined in plugins and used by Librarian::Activate().

**Parameters:**

- *regServices* pointer to instance of RegistrationServices
- *services* pointer to instance of PlatformServices

**typedef Frontend(*)(* PluginManager::fn_frontend_factory)(PlatformServices &services)**

Factory signature for creating Frontends.

**Parameters:**

- *services*
Returns:

typedef Object*(* PluginManager::fn_object_factory)(PlatformServices &services)

Factory signature for creating Objects.

Parameters:
services

Returns:

Enumeration Type Documentation

enum PluginManager::Mode_t

Enum used by serializers for specifying how much of an Object to serializer.

Function Documentation

MAKE_IT_C PLUGIN_API void PluginManager::activate_plugin (RegistrationServices * regServices, PlatformServices * services)

Implemented in every plugin.

Parameters:
regServices A pointer to the platforms registration services. This pointer does not indicate a transfer of ownership but is apparently the only safe way to cross the Library boundary

Here is the call graph for this function:

MAKE_IT_C PLUGIN_API void PluginManager::deactivate_plugin (RegistrationServices * regServices, PlatformServices * services)

Implemented in every plugin.
Parameters:

regServices A pointer to the platforms registration services. This pointer does not indicate a transfer of ownership but is apparently the only safe way to cross the Library boundary.

References

PluginManager::RegistrationServices::DeactivateListener,
PluginManager::PlatformServices::IsOpen,
PluginManager::PlatformServices::Query, and
PluginManager::PlatformServices::signal_object_closed.

MAKE_IT_C PLUGIN_API LibraryData * PluginManager::register_plugin ()

Implemented in every plugin.

Returns:

Information about the plugin

References

PluginManager::LibraryData::LibraryName, PluginManager::LibraryData::LibraryVersion,
PluginManager::Version::Major, PluginManager::Version::Minor,
PluginManager::LibraryData::PluginAPIVersion, PluginManager::LibraryData::ProvidedListeners, and
PluginManager::LibraryData::ProvidedObjects.

---

PluginManager::Exception Namespace Reference

Contains exceptions used by the PluginManager namespace.

Detailed Description

Contains exceptions used by the PluginManager namespace.

Deprecated:

These exceptions will eventually be replaced by exceptions declared using the
FRAMEWORK_DECLARE_EXCEPTION or FRAMEWORK_DECLARE_CHILD_EXCEPTION macros.

---

Utility Namespace Reference

The Utility namespace includes basic functions that do not fit anywhere else.

Classes

- class Converter
- Generic converter class. class Converter< std::string >
- Converter class for std::string return type. class Converter< boost::posix_time::ptime >

Converter class for boost::posix_time::ptime return type. Functions

- const std::string Base64Encode (const std::string &message)
**Detailed Description**

The *Utility* namespace includes basic functions that do not fit anywhere else.

---

**Function Documentation**

`const std::string Utility::Base64Decode (const std::string & message)`

Decodes a message from MIME-style base64 encoding using code Copyright (C) 2004-2008 René Nyffenegger.

**Parameters:**

- `message` the encoded message

**Returns:**

- the original message

`const std::string Utility::Base64Encode (const std::string & message)`

Encodes messages using MIME-style base64 encoding using code Copyright (C) 2004-2008 René Nyffenegger.

**Parameters:**

- `message` the original message

**Returns:**

- the base64 encoded string

Referenced by Insight::HostSynchronizer::Peer::GenerateHostId(), Insight::Directory::Contact::GeneratePrimaryKey(), and DAL::Class::GeneratePrimaryKey().

Here is the caller graph for this function:
template<class outType, class inType> outType Utility::Convert (const inType & in) [inline]

This function template provides automatic conversion between many data types.

The actual conversion process is handled by one of the semi-specialized Converter classes. The current version supports conversion between any two data types for which the input and output stream operators are defined as conversion between std::string and boost::posix_time::ptime

Parameters:  
   in The source value to convert

Returns:  
The converted value in the specified datatype

template<typename T> std::list<T>::iterator Utility::ListFind (std::list<T> & haystack, T & needle) [inline]

Searches an std::list for a specified value.

Parameters:  
   haystack the list to search  
   needle the value to find

Returns:  
an iterator to the found item (or end() if not found)

template<typename T> std::list<T>::const_iterator Utility::ListFind (const std::list<T> & haystack, const T & needle) [inline]

Searches an std::list for a specified value.

Parameters:  
   haystack the list to search  
   needle the value to find

Returns:  
a const_iterator to the found item (or end() if not found)

Referenced by Framework::Registry::Deactivate().

Here is the caller graph for this function:
boost::posix_time::ptime Utility::ParseTimeString (const std::string & timeString)

Parses a string containing a time in any one of a number of formats.

Parameters:
  timeString a string containing a time

Returns:
a boost::posix_time::ptime representation of the time in the timeString.

const std::list< std::string > Utility::Segment (const std::string & string, const std::string & delimiter)

Segments a delimited std::string into a std::list<std::string>

Parameters:
  string The string on which to operator
  delimiter The std::string at which string should be segmented

Returns:
  A list containing the segmented strings

const std::list< std::string > Utility::Segment (const std::string & string, const char & delimiter)

Segments a delimited std::string into a std::list<std::string>

Parameters:
  string The string on which to operator
  delimiter The character at which string should be segmented

Returns:
  A list containing the segmented strings

---

Yoohoo Namespace Reference

The Yoohoo namespace contains the Peer and Session classes the implement connection and discovery components of the Yoohoo protocol.

Classes
- class Peer
  Processes incoming Yoohoo commands. class Session
  Session translates higher level C++ calls into XML for transmission to remote hosts and translates inbound XML from remote hosts into C++ processable signals.
Detailed Description
The Yohoo namespace contains the Peer and Session classes the implement connection and discovery components of the Yohoo protocol.

Class Documentation

**Insight::Communication::Account Class Reference**

Defines an interface from which communication accounts (e.g. POP3, SMTP, etc) can be derived.
#include <Account.hpp>
Inherits PluginManager::Object.
Inherited by Insight::Communication::DummyAccount, Insight::POP3::Account, Insight::RSS::Account, and Insight::SMTP::Account.
Collaboration diagram for Insight::Communication::Account:
Public Member Functions

- **virtual ~Account ()**
  Closes related Messages.

- **virtual const std::string GetHost () const =0**
  Returns the host related to this account. This is a largely a hack to make InternetMessageFormat::GenerateMessageID work.

Protected Attributes

- **DAL::Field<int> UpdateInterval**

Detailed Description

Defines an interface from which communication accounts (e.g. POP3, SMTP, etc) can be derived.

Constructor & Destructor Documentation

**Insight::Communication::Account::~Account () [virtual]**

Closes related Messages.
References DAL::Class::GetPrimaryKey().
Here is the call graph for this function:

```
Insight::Communication::Account::~Account  
DAL::Class::GetPrimaryKey
```

Member Function Documentation

**virtual const std::string Insight::Communication::Account::GetHost () const [pure virtual]**

Returns the host related to this account. This is a largely a hack to make InternetMessageFormat::GenerateMessageID work.

Implemented in **Insight::Communication::DummyAccount** (p.105), **Insight::POP3::Account** (p.66), **Insight::RSS::Account** (p.70), and **Insight::SMTP::Account** (p.73).

Referenced by Insight::InternetMessageFormat::Message::GenerateMessageId().
Here is the caller graph for this function:

```
Insight::Communication::Account::GetHost  
Insight::InternetMessageFormat::Message::GenerateMessageId
```
Member Data Documentation

DAL::Field<int> Insight::Communication::Account::UpdateInterval [protected]
   This should be considered a value in minutes

The documentation for this class was generated from the following files:
- src/Communication/Account.hpp
- src/Communication/Account.cpp

Insight::POP3::Account Class Reference

Contains the functions necessary to retrieve mail via the POP3 protocol.
#include <Account.hpp>
Inherits Insight::Communication::Account, and Insight::Communication::Receiver.
Collaboration diagram for Insight::POP3::Account:
Public Member Functions

- virtual const std::string GetHost() const
  Returns the host related to this account. This is a largely a hack to make InternetMessageFormat::GenerateMessageID work.

- const std::string & GetUsername() const
- const std::string & GetPassword() const
- const std::string & GetServer() const
- const int & GetPort() const
- const Communication::Protocol::SecurityTypeLookup & GetSecurityType() const

Indicates whether or not this client will delete messages from the server after retrieving them.

- const bool & GetLeaveOnServer() const
- const bool & GetHeadersOnly() const
  Indicates whether or not this client will download the full messages or just the message header.

Static Public Member Functions

- static PluginManager::Object * Factory (PluginManager::PlatformServices &services)
  Factory function.

Detailed Description
Contains the functions necessary to retrieve mail via the POP3 protocol.

Member Function Documentation

PluginManager::Object * Insight::POP3::Account::Factory (PluginManager::PlatformServices & services) [static]

Factory function.

Parameters:
  services

Returns:
  a pointer to the newly allocated Account

const bool & Insight::POP3::Account::GetHeadersOnly() const

Indicates whether or not this client will download the full messages or just the message header.

Returns:
  true if only headers will be downloaded, false otherwise
const std::string & Insight::POP3::Account::GetHost () const [virtual]

Returns the host related to this account. This is a largely a hack to make InternetMessageFormat::GenerateMessageID work.
Implements Insight::Communication::Account (p.62).

const bool & Insight::POP3::Account::GetLeaveOnServer () const

Indicates whether or not this client will delete messages from the server after retrieving them.

Returns:
true if messages will be left on server, false otherwise.

const std::string & Insight::POP3::Account::GetPassword () const

Returns:
The POP3 password

const int & Insight::POP3::Account::GetPort () const

Returns:
The POP3 server port

const Communication::Protocol::SecurityTypeLookup & Insight::POP3::Account::GetSecurityType () const

Returns:
The security type in use

const std::string & Insight::POP3::Account::GetServer () const

Returns:
The POP3 server name

const std::string & Insight::POP3::Account::GetUsername () const

Returns:
The POP3 username

The documentation for this class was generated from the following files:
- plugin_pop3/Account.hpp
- plugin_pop3/Account.cpp
Insight::RSS::Account Class Reference

defines an RSS Account
#include <Account.hpp>
Inherits Insight::Communication::Account, and Insight::Communication::Receiver.
Collaboration diagram for Insight::RSS::Account:
Public Member Functions

- virtual void GeneratePrimaryKey()
  Generates a primary key for the Object.

- virtual const std::string GetHost() const
  Returns the host related to this account. This is a largely a hack to make InternetMessageFormat::GenerateMessageID work.

Static Public Member Functions

- static PluginManager::Object * Factory(PluginManager::PlatformServices &services)

Protected Member Functions

- Account(PluginManager::PlatformServices &services)
  Constructor. Sets defaults.

Detailed Description

defines an RSS Account

Constructor & Destructor Documentation

Insight::RSS::Account::Account (PluginManager::PlatformServices & services) [protected]

Constructor. Sets defaults.

Parameters:

services
References DAL::Class::Add(), and Insight::Communication::Receiver::signal_message_received.
Referenced by Factory().

Here is the call graph for this function:

Here is the caller graph for this function:
Member Function Documentation

PluginManager::Object * Insight::RSS::Account::Factory (PluginManager::PlatformServices & services) [static]


Parameters:
services

Returns:
a pointer to the new Account

References Account().

Here is the call graph for this function:

```
Insight::RSS::Account::Factory             -> Insight::RSS::Account::Account          -> DAL::Class::Add
```

void Insight::RSS::Account::GeneratePrimaryKey () [virtual]

Generates a primary key for the Object.

This calls Class::GeneratePrimaryKey, but adds the additional procedure of firing PlatformServices::signal_primaryKey_changed

Reimplemented from PluginManager::Object (p.174).

const std::string Insight::RSS::Account::GetHost () const [virtual]

Returns the host related to this account. This is a largely a hack to make InternetMessageFormat::GenerateMessageID work.

Implements Insight::Communication::Account (p.62).

The documentation for this class was generated from the following files:

- plugin_rss/Account.hpp
- plugin_rss/Account.cpp

Insight::SMTP::Account Class Reference

uses SMTP::Session to send Internet Messages

#include <Account.hpp>

Inherits Insight::Communication::Account, and Insight::Communication::Sender.

Collaboration diagram for Insight::SMTP::Account:
Public Member Functions

- virtual const std::string GetHost() const
  
  Returns the host related to this account. This is a largely a hack to make InternetMessageFormat::GenerateMessageID work.

- Account (PluginManager::PlatformServices &services)
  
  Constructor. Sets some sensible defaults for SMTP parameters.

  Parameters:
  services
  References DAL::Class::Add().
  Referenced by Factory().

  Here is the call graph for this function:

  ![Call Graph for Account Constructor](image)

  Here is the caller graph for this function:

  ![Caller Graph for Account Constructor](image)

Static Public Member Functions

- static PluginManager::Object * Factory (PluginManager::PlatformServices &services)
  

Detailed Description

uses SMTP::Session to send Internet Messages

Constructor & Destructor Documentation

Insight::SMTP::Account::Account (PluginManager::PlatformServices & services)

Constructor. Sets some sensible defaults for SMTP parameters.

Parameters:

services

References DAL::Class::Add().
Referenced by Factory().

Here is the call graph for this function:

![Call Graph for Account Constructor](image)

Here is the caller graph for this function:

![Caller Graph for Account Constructor](image)
Member Function Documentation

PluginManager::Object * Insight::SMTP::Account::Factory (PluginManager::PlatformServices & services) [static]


Parameters:

services

Returns:

a pointer to the account

References Account().

Here is the call graph for this function:

const std::string & Insight::SMTP::Account::GetEmailAddress () const

Returns:

The email address associated with this account

const std::string Insight::SMTP::Account::GetHost () const [virtual]

Returns the host related to this account. This is a largely a hack to make
InternetMessageFormat::GenerateMessageID work.

Implements Insight::Communication::Account (p.62).

const std::string & Insight::SMTP::Account::GetPassword () const

Returns:

The SMTP password

const int & Insight::SMTP::Account::GetPort () const

Returns:

The SMTP Server port

const bool & Insight::SMTP::Account::GetRequireAuthentication () const

Returns:

A bool indicating whether not the client will attempt to authenticate before sending mail
const Communication::Protocol::SecurityTypeLookup & Insight::SMTP::Account::GetSecurityType () const

Returns:
The Security type in use

const std::string & Insight::SMTP::Account::GetServer () const

Returns:
The SMTP Server name

const std::string & Insight::SMTP::Account::GetUsername () const

Returns:
The SMTP user name

The documentation for this class was generated from the following files:
- plugin_smtp/src/Account.hpp
- plugin_smtp/src/Account.cpp

PluginManager::Backend Class Reference

This class handles the storage and retrieval of data.
#include <Backend.hpp>
Inherited by Insight::MySQL::Backend.
Collaboration diagram for PluginManager::Backend:
```plaintext
PluginManager::Version
+ Major
+ Minor
+ Version()
+ Version()

PluginApi::version

PluginManager::PlatformServices
+ PluginApi::version
+ MakeObject
+ MakeKeyedObject
+ GetObject
+ PutObject
+ DeleteObject
+ isOpen
+ GetBackend
+ GetCurrent
+ GetListener
+ InsertLibrary
+ EjectLibrary
+ ListLibraryStatus
+ ListLibraryDetails
+ signal_user_alert
+ signal_user_abort_offer
+ signal_user_confirm_request
+ signal_user_text_request
+ signal_object_committed
+ signal_object_changed
+ signal_object_not_changed
+ signal_object_updated
+ signal_object_deleted
+ signal_object_delete_pending
+ signal_object_closed
+ signal_primary_key_changed
+ ListBackends
+ ListListeners
+ ListObjectTypes
+ ListObjectSupertypes
+ ListSerializers
+ Serialize
+ SerializeToFile
+ Deserialize
+ DeserializeFromFile
+ signal_library_loaded
+ signal_plugin_registered
+ signal_plugin_activated
+ signal_library_unloaded
+ signal_plugin_unregistered
+ signal_plugin_deactivated
+ LogEvent
+ SetPreference
+ GetPreference
+ signal_preference_changed
+ Query
+ Exec

+ PlatformServices()

Services

PluginManager::Backend
# Services
+ Backend()
+ LoadObject()
+ SaveObject()
+ DeleteObject()
+ Query()
+ Exec()
+ Define()
+ Knows()
# handle_object_deleted()
# handle_primary_key_changed()
```
Public Types

- `typedef std::vector<std::map<std::string, std::string>> QueryResult`
  
  This a vector of string-keyed, string-valued maps used for accessing the data returned by a database query.

Public Member Functions

- **Backend (PlatformServices &services)**
  Constructor.

- **virtual Object & LoadObject (const std::string &primaryKey)=0**
  Loads an Object from the database.

- **virtual void SaveObject (Object &object)=0**
  Saves an Object to the database.

- **virtual void DeleteObject (const std::string &primaryKey)=0**
  Deletes an Object from the database.

- **virtual QueryResult Query (const std::string &sql)=0**
  Executes an SQL statement that expects a result.

- **virtual void Exec (const std::string &sql)=0**
  Executes an SQL statement that does not expect a result.

- **virtual void Define (DAL::Class &c)=0**
  Generates the rules necessary to maintain data integrity.

- **virtual bool Exists (const std::string &primaryKey)=0**

---

Detailed Description

This class handles the storage and retrieval of data.

---

Member Typedef Documentation

**typedef std::vector<std::map<std::string, std::string>> > PluginManager::Backend::QueryResult**

This a vector of string-keyed, string-valued maps used for accessing the data returned by a database query.

---

Constructor & Destructor Documentation

**PluginManager::Backend::Backend (PlatformServices & services)**

Constructor.

**Parameters:**

- **services**
  Reimplemented in Insight::MySQL::Backend (p.80).
Member Function Documentation

virtual void PluginManager::Backend::Define (DAL::Class & c) [pure virtual]

Generates the rules necessary to maintain data integrity.
For example, this function creates the SQL CREATE TABLE scripts to define the database

Parameters:
   c The class from which to create scripts
Implemented in Insight::MySQL::Backend (p.81).

virtual void PluginManager::Backend::DeleteObject (const std::string & primaryKey) [pure virtual]

Deletes an Object from the database.

Parameters:
   primaryKey
Implemented in Insight::MySQL::Backend (p.81).

virtual void PluginManager::Backend::Exec (const std::string & sql) [pure virtual]

Executes an SQL statement that does not expect a result.

Parameters:
   sql The SQL statement to be executed
Implemented in Insight::MySQL::Backend (p.81).

virtual bool PluginManager::Backend::Exists (const std::string & primaryKey) [pure virtual]

Determines whether or not an Object with the specified primary key exists in the database without
loading the entire Object into memory

Parameters:
   primaryKey

Returns:
   true if it exists, false otherwise
Implemented in Insight::MySQL::Backend (p.81).

virtual Object& PluginManager::Backend::LoadObject (const std::string & primaryKey) [pure virtual]

Loads an Object from the database.
Parameters:

primaryKey the primary key of the object to a load

Returns:

A reference to the newly loaded Object

Implemented in Insight::MySQL::Backend (p. 82).

virtual QueryResult PluginManager::Backend::Query (const std::string & sql) [pure virtual]

Executes an SQL statement that expects a result.

Parameters:

sql The SQL query to be run

Returns:

A query result containing the results of the query. Note: As QueryResult is currently configured, this function will not return the full result if there are column name collisions in a multiple table query.

Implemented in Insight::MySQL::Backend (p. 82).

virtual void PluginManager::Backend::SaveObject (Object & object) [pure virtual]

Saves an Object to the database.

Parameters:

object The Object to save

Implemented in Insight::MySQL::Backend (p. 82).

The documentation for this class was generated from the following files:

- insight_pluginmanager/src/Backend.hpp
- insight_pluginmanager/src/Backend.cpp

Insight::MySQL::Backend Class Reference

Adapts MySQL++ to the PluginManager::Backend interface.

#include <Backend.hpp>

Inherits PluginManager::Backend.

Collaboration diagram for Insight::MySQL::Backend:
Public Member Functions

- **Backend** (PluginManager::PlatformServices &services)
  Constructor.
- virtual **PluginManager::Object** & **LoadObject** (const std::string &primaryKey)
  Loads an Object from the database.
- virtual void **SaveObject** (PluginManager::Object &object)
  Saves an Object to the database.
- virtual void **DeleteObject** (const std::string &primaryKey)
  Deletes an Object from the database.
- virtual PluginManager::Backend::QueryResult **Query** (const std::string &sql)
  Executes an SQL statement that expects a result.
- virtual void **Exec** (const std::string &sql)
  Executes an SQL statement that does not expect a result.
- virtual bool **Exists** (const std::string &primaryKey)
- virtual void **Define** (DAL::Class &c)
  Generates the rules necessary to maintain data integrity.

Protected Member Functions

- virtual void **handle_object_deleted** (const std::string &primaryKey)

---

Detailed Description

Adapts MySQL++ to the PluginManager::Backend interface.

---

Constructor & Destructor Documentation

**Insight::MySQL::Backend::Backend** (PluginManager::PlatformServices & services)

Constructor.

**Parameters:**

- **services**
  Reimplemented from PluginManager::Backend (p.76).
  References Exec(), and Query().

Here is the call graph for this function:

```
Insight::MySQL::Backend::Backend
  └── Insight::MySQL::Backend::Exec
  └── Insight::MySQL::Backend::Query
```
Member Function Documentation

void Insight::MySQL::Backend::Define (DAL::Class & c) [virtual]

Generates the rules necessary to maintain data integrity.
For example, this function creates the SQL CREATE TABLE scripts to define the database

Parameters:

- c The class from which to create scripts

Implements PluginManager::Backend (p.77).

virtual void Insight::MySQL::Backend::DeleteObject (const std::string & primaryKey) [virtual]

Deletes an Object from the database.

Parameters:

- primaryKey

Implements PluginManager::Backend (p.77).

void Insight::MySQL::Backend::Exec (const std::string & sql) [virtual]

Executes an SQL statement that does not expect a result.

Parameters:

- sql The SQL statement to be executed

Implements PluginManager::Backend (p.77).

Referenced by Backend().

Here is the caller graph for this function:

![Caller Graph]

bool Insight::MySQL::Backend::Exists (const std::string & primaryKey) [virtual]

Determines whether or not an Object with the specified primary key exists in the database without loading the entire Object into memory

Parameters:

- primaryKey

Returns:

- true if it exists, false otherwise

Implements PluginManager::Backend (p.77).

References Query().

Here is the call graph for this function:

![Call Graph]
void Insight::MySQL::Backend::handle_object_deleted (const std::string & primaryKey)  
[protected, virtual]  
Records the time at which Objects are deleted.

Parameters:

primaryKey
Implements PluginManager::Backend (p.74).

PluginManager::Object & Insight::MySQL::Backend::LoadObject (const std::string & primaryKey)  
[virtual]

Loads an Object from the database.

Parameters:

primaryKey the primary key of the object to a load

Returns:

A reference to the newly loaded Object
Implements PluginManager::Backend (p.77).

PluginManager::Backend::QueryResult Insight::MySQL::Backend::Query (const std::string & sql)  
[virtual]

Executes an SQL statement that expects a result.

Parameters:

sql The SQL query to be run

Returns:

A query result containing the results of the query. Note: As QueryResult is currently configured, this
function will not return the full result if there are column name collisions in a multiple table query.
Implements PluginManager::Backend (p.78).
Referenced by Backend(), and Exists().

Here is the caller graph for this function:

virtual void Insight::MySQL::Backend::SaveObject (PluginManager::Object & object)  
[virtual]

Saves an Object to the database.
Parameters:

object The Object to save

Implements PluginManager::Backend (p. 78).

The documentation for this class was generated from the following files:

- insight_plugin_mysql/src/Backend.hpp
- insight_plugin_mysql/src/Backend.cpp

Framework::Managers::BackendManager Class Reference

manages the available backends

#include <BackendManager.hpp>

Collaboration diagram for Framework::Managers::BackendManager:
Public Member Functions

- **BackendManager** (Registry &registry, PluginManager::PlatformServices &services, PluginManager::RegistrationServices &regServices)
  Constructor. Binds functions and adds Options.

- virtual ~BackendManager ()
  Destructor. Deactivates backends and factories.

- virtual PluginManager::Backend & Get (const std::string &name)
  Retrieves the specified backend.

Protected Member Functions

- virtual void Activate (PluginManager::fn_backend_factory factory, const std::string &name)

Detailed Description

manages the available backends

Constructor & Destructor Documentation

Framework::Managers::BackendManager::BackendManager (Registry & registry, PluginManager::PlatformServices & services, PluginManager::RegistrationServices & regServices)

Constructor. Binds functions and adds Options.

Parameters:

registry
services
regServices

Returns:

References PluginManager::RegistrationServices::ActivateBackendFactory, PluginManager::RegistrationServices::DeactivateBackendFactory, PluginManager::PlatformServices::GetBackend, and Framework::Librarian::GetOptionDescriptions().

Here is the call graph for this function:

Framework::Managers::BackendManager::Get
Framework::Managers::BackendManager::~BackendManager () [virtual]

Destructor. Deactivates backends and factories.
Member Function Documentation

```cpp
void Framework::Managers::BackendManager::Activate (PluginManager::fn_backend_factory factory, const std::string & name) [protected, virtual]

Used by RegistrationServices

Parameters:

factory
name

PluginManager::Backend & Framework::Managers::BackendManager::Get (const std::string & name) [virtual]

Retrieves the specified backend.

Parameters:

name The name of the backend to retrieve

Returns:

The specified backend

References Framework::Librarian::GetOptions().

Referenced by BackendManager().

Here is the call graph for this function:

```
Framework::Managers::BackendManager::Get  ---> Framework::Librarian::GetOptions
```

Here is the caller graph for this function:

```
Framework::Managers::BackendManager::Get  ---> Framework::Managers::BackendManager::BackendManager
```

The documentation for this class was generated from the following files:

- BackendManager.hpp
- BackendManager.cpp

---

**DAL::Class Class Reference**

A class represents a table (or collection of tables) within a database.

```cpp
#include <Class.hpp>
```

Inherited by Insight::Directory::Address, and **PluginManager::Object**.

**Public Member Functions**

- **Class** (const std::string &group)
- **virtual const std::string & GetPrimaryKey () const**
virtual bool IsAltered() const
virtual const std::string & GetGroup() const

Protected Member Functions
virtual void Add (const std::string &group, FieldBase &fb)
virtual void Add (const std::string &group, Collection &collection)
virtual void GeneratePrimaryKey()

Detailed Description
A class represents a table (or collection of tables) within a database.

Constructor & Destructor Documentation

DAL::Class::Class (const std::string & group)
Constructor. Calls GeneratePrimaryKey
Parameters:
group The name of table in which the Class’s fields are stored
References GeneratePrimaryKey().

Here is the call graph for this function:

Member Function Documentation

void DAL::Class::Add (const std::string & group, Collection & collection) [protected, virtual]
Used by derived classes to register their Collections with the Class
Parameters:
group The group (table) to which the field belongs
collection A reference to the Collection to add
References DAL::Collection::GetGroup().
Here is the call graph for this function:

void DAL::Class::Add (const std::string & group, FieldBase & fb) [protected, virtual]
Used by derived classes to register their Fields with the Class
Parameters:
group The group (table) to which the field belongs
fb A reference to the Field to add
void DAL::Class::GeneratePrimarykey () [protected, virtual]

Creates PrimaryKey. Can be overridden in derived classes. There may be a few cases in which the primary key should be changed after constructions, but before being saved.

The base implementation simply sets the current time as the PrimaryKey

Reimplemented in Insight::Directory::Contact (p.99), Insight::InternetMessageFormat::Message (p.166), Insight::RSS::Account (p.70), Insight::RSS::Item (p.136), and PluginManager::Object (p.174).

Referenced by Insight::SMTP::Account::Account(), and Insight::RSS::Account::Account(), and Insight::RSS::Item::Item().

Here is the caller graph for this function:

const std::string & DAL::Class::GetGroup () const [virtual]

Returns:
The name of group (table) for this Class

const std::string & DAL::Class::GetPrimaryKey () const [virtual]

Returns this instance's PRIMARY KEY

Returns:
The value of this instance's PRIMARY KEY

Referenced by Insight::InternetMessageFormat::Message::GenerateMessageId(), DAL::Reference::operator=(), and Insight::Communication::Account::~Account().

Here is the caller graph for this function:
bool DAL::Class::IsAltered () const [virtual]
Indicates whether any of the Class’s parameters have been changed since the last database interaction

The documentation for this class was generated from the following files:
- Class.hpp
- Class.cpp

DAL::Collection Class Reference

The Collection class provides an interface for working with multiple values in the same variable.
#include <Collection.hpp>
Inherited by DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >, DAL::STLCollection< std::map< KEY_TYPE, VAL_TYPE >, std::pair< const KEY_TYPE, VAL_TYPE >, const std::pair< const KEY_TYPE, VAL_TYPE > >, DAL::STLCollection< std::map< std::string, PhoneNumber >, std::pair< const std::string, PhoneNumber >, const std::pair< const std::string, PhoneNumber > >, and DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, const std::pair< const std::string, std::string > >.
Collaboration diagram for DAL::Collection:
Public Member Functions

- **Collection** (const std::string &group, const Class &c)
  Constructor.
- **Collection** (const Collection &other)
- virtual void **for_each** (boost::function< void(FieldMap &fields)> functor)=0
- virtual void **for_each** (boost::function< void(ConstFieldMap &fields)> functor) const =0
- virtual const Reference & **GetForeignKey** () const
  Gets the ForeignKey to which this instance is related.
- virtual const std::string & **GetGroup** () const
  Gets this instance's group.
- virtual bool **IsAltered** () const =0
  Indicates if any of Collections fields have been changed since the last database interaction.
- virtual void **SetAltered** (const bool &altered)=0
  Can be used to explicitly override the altered state (mostly used by Transactions after they have done their work.
- virtual void **Rekey** (const std::string &key)
  Called when the instance's foreign key is changed.
- virtual bool **empty** () const =0
  STL Wrapper Function.
- virtual size_t **size** () const =0
  STL Wrapper Function.
- virtual size_t **max_size** () const =0
  STL Wrapper Function.
- virtual void **clear** ()=0
  STL Wrapper Function.
- virtual void **InsertDummy** ()=0
  Inserts a dummy entry for schema generation.

Protected Attributes

- std::string **Group**

Detailed Description

The **Collection** class provides an interface for working with multiple values in the same variable.

The C++ Standard Template Library provides container classes such as std::list and std::map for storing multiple values in one variable. Unfortunately, these containers do not immediately lend themselves to database interactions. **Collection** is an abstract class that provides many of the same features as **FieldBase**.

Constructor & Destructor Documentation

DAL::Collection::Collection (const std::string &group, const Class & c)
Constructor.

**Parameters:**
- **group** This is effectively the name of the table in which this Collection's values will be stored.
- **c** This is the Class instance to which this Collection instance is related.

**DAL::Collection::Collection (const Collection & other)**

Copy Constructor

**Parameters:**
- **other** The instance from which to copy data

**Member Function Documentation**

**virtual void DAL::Collection::clear () [pure virtual]**

STL Wrapper Function. *

Implemented in DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE > (p.297), DAL::STLCollection< std::map< KEY_TYPE, VAL_TYPE >, std::pair< const KEY_TYPE, VAL_TYPE >, const std::pair< const KEY_TYPE, VAL_TYPE > > (p.297), DAL::STLCollection< std::map< std::string, PhoneNumber >, std::pair< const std::string, PhoneNumber >, const std::pair< const std::string, PhoneNumber > > (p.297), and DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, const std::pair< const std::string, std::string > > (p.297).

**virtual bool DAL::Collection::empty () const [pure virtual]**

STL Wrapper Function.

**Returns:**
true if there are no items in the Collection, false otherwise.

Implemented in DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE > (p.297), DAL::STLCollection< std::map< KEY_TYPE, VAL_TYPE >, std::pair< const KEY_TYPE, VAL_TYPE >, const std::pair< const KEY_TYPE, VAL_TYPE > > (p.297), DAL::STLCollection< std::map< std::string, PhoneNumber >, std::pair< const std::string, PhoneNumber >, const std::pair< const std::string, PhoneNumber > > (p.297), and DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, const std::pair< const std::string, std::string > > (p.297).

**virtual void DAL::Collection::for_each (boost::function< void(ConstFieldMap &fields)> functor) const [pure virtual]**

Iterates of the items in the Collection and calls functor on the each item. A dereferenced Collection iterator (first defined in STLCollection) returns FieldMap.

**Parameters:**
- **functor** A boost::function object that can be called on each item in the Collection
virtual void DAL::Collection::for_each (boost::function< void(FieldMap &fields)> functor) [pure virtual]

Iterates of the items in the Collection and calls functor on the each item. A dereferenced Collection iterator (first defined in STLCollection) returns FieldMap.

Parameters:
functor A boost::function object that can be called on each item in the Collection.

const Reference & DAL::Collection::GetForeignKey () const [virtual]

Gets the ForeignKey to which this instance is related.

Returns:
This instance's ForeignKey

const std::string & DAL::Collection::GetGroup () const [virtual]

Gets this instance's group.

Returns:
The name of this instance's group

References Group.

Referenced by DAL::Class::Add().

Here is the caller graph for this function:

virtual void DAL::Collection::InsertDummy () [pure virtual]

Inserts a dummy entry for schema generation.
Implemented in DAL::Collections::Map< KEY_TYPE, VAL_TYPE > (p.162), DAL::Collections::ReferenceSequence (p.230), DAL::Collections::Sequence< VAL_TYPE > (p.251), DAL::Collections::Map< std::string, PhoneNumber > (p.162), DAL::Collections::Map< std::string, std::string > (p.162), DAL::Collections::Sequence< std::string > (p.251), and DAL::Collections::Sequence< Reference > (p.251).

**virtual bool DAL::Collection::IsAltered () const [pure virtual]**

Indicates if any of Collections fields have been changed since the last database interaction.

**Returns:**

true if an alteration has occurred, false otherwise

Implemented in DAL::Collections::Map< KEY_TYPE, VAL_TYPE > (p.162), DAL::Collections::Sequence< VAL_TYPE > (p.251), DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE > (p.299), DAL::Collections::Map< std::string, PhoneNumber > (p.162), DAL::Collections::Map< std::string, std::string > (p.162), DAL::Collections::Sequence< std::string > (p.251), DAL::Collections::Sequence< Reference > (p.251), DAL::STLCollection< std::map< KEY_TYPE, VAL_TYPE >, std::pair< const KEY_TYPE, VAL_TYPE >, const std::pair< const KEY_TYPE, VAL_TYPE > > (p.299), DAL::STLCollection< std::map< std::string, PhoneNumber >, std::pair< const std::string, PhoneNumber >, const std::pair< const std::string, PhoneNumber > > (p.299), and DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, const std::pair< const std::string, std::string > > (p.299).

**virtual size_t DAL::Collection::max_size () const [pure virtual]**

STL Wraper Function.

**Returns:**

The maximum number of items that the Collection can store

Implemented in DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE > (p.299), DAL::STLCollection< std::map< KEY_TYPE, VAL_TYPE >, std::pair< const KEY_TYPE, VAL_TYPE >, const std::pair< const KEY_TYPE, VAL_TYPE > > (p.299), DAL::STLCollection< std::map< std::string, PhoneNumber >, std::pair< const std::string, PhoneNumber >, const std::pair< const std::string, PhoneNumber > > (p.299), and DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, const std::pair< const std::string, std::string > > (p.299).

**void DAL::Collection::Rekey (const std::string & key) [virtual]**

Called when the instance's foreign key is changed.

**Parameters:**

key the new key value

References DAL::Reference::Rekey().

Here is the call graph for this function:

![Call Graph](https://example.com/call_graph.png)
virtual void DAL::Collection::SetAltered (const bool & altered) [pure virtual]

Can be used to explicitly override the altered state (mostly used by Transactions after they have done their work.

**Parameters:**

*altered* The new value for Altered

Implemented in DAL::Collections::Map< KEY_TYPE, VAL_TYPE > (p.163), DAL::Collections::Sequence< VAL_TYPE > (p.252), DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE > (p.300), DAL::Collections::Map< std::string, PhoneNumber > (p.163), DAL::Collections::Map< std::string, std::string > (p.163), DAL::Collections::Sequence< std::string > (p.252), DAL::Collections::Sequence< Reference > (p.252), DAL::STLCollection< std::map< KEY_TYPE, VAL_TYPE >, std::pair< const KEY_TYPE, VAL_TYPE >, std::pair< const KEY_TYPE, VAL_TYPE > > (p.300), and DAL::STLCollection< std::map< std::string, PhoneNumber >, std::pair< const std::string, PhoneNumber >, const std::pair< const std::string, PhoneNumber > > (p.300), and DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, std::pair< const std::string, std::string > > (p.300).

virtual size_t DAL::Collection::size () const [pure virtual]

STL Wrapper Function.

**Returns:**

the number of items stored in the Collection

Implemented in DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE > (p.301), DAL::STLCollection< std::map< KEY_TYPE, VAL_TYPE >, std::pair< const KEY_TYPE, VAL_TYPE >, std::pair< const KEY_TYPE, VAL_TYPE > > (p.301), DAL::STLCollection< std::map< std::string, PhoneNumber >, std::pair< const std::string, PhoneNumber >, const std::pair< const std::string, PhoneNumber > > (p.301), and DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, std::pair< const std::string, std::string > > (p.301).

---

**Member Data Documentation**

std::string DAL::Collection::Group [protected]

e.g. table name

Referenced by GetGroup().

---

The documentation for this class was generated from the following files:

- Collection.hpp
- Collection.cpp
NetworkInterface::Connection Class Reference

#include <Connection.hpp>
Inherited by NetworkInterface::PlainTextConnection, and NetworkInterface::SslConnection.

Public Member Functions

- virtual ~Connection ()
  Disconnects the connection if it is still connected.
- virtual const std::string ReadUntil (const std::string &endText)=0
  Listens until the specified text is encountered.
- virtual void Write (const std::string &text)=0

Detailed Description

A Connection is generic interface for working with a remote host. It's functions are implemented by other classes that need not be exposed to the API.

Constructor & Destructor Documentation

NetworkInterface::Connection::~Connection () [virtual]

Disconnects the connection if it is still connected.

Member Function Documentation

virtual const std::string NetworkInterface::Connection::ReadUntil (const std::string & endText) [pure virtual]

Listens until the specified text is encountered.

Parameters:
  endText the delimiter to listen for.

Returns:
  The desired string

virtual void NetworkInterface::Connection::Write (const std::string & text) [pure virtual]

Sends text to the other end of the Connection

Parameters:
  text the text to send
The documentation for this class was generated from the following files:
- Connection.hpp
- Connection.cpp

Insight::Directory::Contact Class Reference

Defines an Object for storing Contact information.
#include <Contact.hpp>
Inherits PluginManager::Object.
Inherited by Insight::Directory::DummyContact, and Insight::Directory::Person.
Collaboration diagram for Insight::Directory::Contact:
Public Member Functions

- virtual const std::string ToMailbox () const =0
  represents a Contact as an RFC 2821 compatible mailbox
- virtual void GeneratePrimaryKey ()
  Generates a primary key based on the Contact's email address.

Detailed Description

Defines an Object for storing Contact information.

Member Function Documentation

void Insight::Directory::Contact::GeneratePrimaryKey () [virtual]

Generates a primary key based on the Contact's email address.
The Primary Key is a base64 encoded representation of the Contact's email address
Reimplemented from PluginManager::Object (p.174).
References Utility::Base64Encode(), and PluginManager::Object::NewKeyAtInsert.
Here is the call graph for this function:

```
Insight::Directory::Contact::GeneratePrimaryKey  -->  Utility::Base64Encode
```

virtual const std::string Insight::Directory::Contact::ToMailbox () const [pure virtual]

represents a Contact as an RFC 2821 compatible mailbox

Returns:

Implemented in Insight::Directory::DummyContact (p.107), Insight::Directory::Person (p.195), and
Insight::Directory::DummyPerson (p.111).

The documentation for this class was generated from the following files:

- Contact.hpp
- Contact.cpp
Utility::Converter< outType > Class Template Reference

Generic converter class.
#include <Convert.hpp>

Detailed Description

\texttt{template<class\ outType>\ class\ Utility::Converter<\ outType>}

Generic converter class.
Defines a generic conversion function that can convert between any two data types for which the input and output stream operators are defined

The documentation for this class was generated from the following file:
- Convert.hpp

Utility::Converter< boost::posix_time::ptime > Class Template Reference

Converter class for boost::posix_time::ptime return type.
#include <Convert.hpp>

Detailed Description

\texttt{template<>\ class\ Utility::Converter<\ boost::posix_time::ptime>}

Converter class for boost::posix_time::ptime return type.
Uses \texttt{Utility::ParseTimeString} to parse various forms string representations of date and time values and convert them to boost::posix_time::ptime

The documentation for this class was generated from the following file:
- Convert.hpp

Utility::Converter< std::string > Class Template Reference

Converter class for std::string return type.
#include <Convert.hpp>
Detailed Description

template<> class Utility::Converter< std::string >

Converter class for std::string return type.
This version is almost identical to the generic form, but defines a slight shortcut used when return std::string. Also, if the input is a boost::posix_time::ptime, calls the boost::posix_time::to_iso_extended_string function on it.

The documentation for this class was generated from the following file:
• Convert.hpp

Insight::MySQL::Transactions::Define Class Reference
	his is a functor that generates make table scripts from Objects
#include <Define.hpp>
Inherits Insight::MySQL::SQLTransaction.
Collaboration diagram for Insight::MySQL::Transactions::Define:
**Detailed Description**

this is a functor that generates make table scripts from Objects

The documentation for this class was generated from the following files:

- Define.hpp
- Define.cpp

---

**Insight::Communication::DummyAccount Class Reference**

```
#include <Account.hpp>
```

Inherits `Insight::Communication::Account`.

Collaboration diagram for `Insight::Communication::DummyAccount`:
Public Member Functions

- virtual const std::string GetHost () const
  
  Returns the host related to this account. This is a largely a hack to make
  InternetMessageFormat::GenerateMessageID work.

Detailed Description

Used by DefinePlatform()

Member Function Documentation

virtual const std::string Insight::Communication::DummyAccount::GetHost () const [inline, virtual]

Returns the host related to this account. This is a largely a hack to make
InternetMessageFormat::GenerateMessageID work.

Implements Insight::Communication::Account (p.62).

The documentation for this class was generated from the following file:
- src/Communication/Account.hpp

Insight::Directory::DummyContact Class Reference

#include <Contact.hpp>

Inherits Insight::Directory::Contact.

Collaboration diagram for Insight::Directory::DummyContact:
Public Member Functions

- virtual const std::string ToMailbox () const
  represents a Contact as an RFC 2821 compatible mailbox

Detailed Description

Used by DefinePlatform()

Member Function Documentation

virtual const std::string Insight::Directory::DummyContact::ToMailbox () const [inline, virtual]

  represents a Contact as an RFC 2821 compatible mailbox

Returns:

Implements Insight::Directory::Contact (p.99).

The documentation for this class was generated from the following file:
- Contact.hpp

Insight::Communication::DummyMessage Class Reference

#include <Message.hpp>
Inherits Insight::Communication::Message.
Collaboration diagram for Insight::Communication::DummyMessage:
Public Member Functions

- virtual const std::string ToNativeFormat () const

Detailed Description

Used by DefinePlatform()

Member Function Documentation

virtual const std::string Insight::Communication::DummyMessage::ToNativeFormat () const
[inline, virtual]

Formats a message as it would appear when downloaded.
Implements Insight::Communication::Message (p.170).

The documentation for this class was generated from the following file:
- src/Communication/Message.hpp

Insight::Directory::DummyPerson Class Reference

#include <Person.hpp>
Inherits Insight::Directory::Person.
Collaboration diagram for Insight::Directory::DummyPerson:
Public Member Functions

- virtual const std::string ToMailbox () const
  represents a Contact as an RFC 2821 compatible mailbox

Detailed Description

Used by DefinePlatform()

Member Function Documentation

virtual const std::string Insight::Directory::DummyPerson::ToMailbox () const [inline, virtual]

  represents a Contact as an RFC 2821 compatible mailbox

Returns:

Reimplemented from Insight::Directory::Person (p.195).

The documentation for this class was generated from the following file:
- src/Directory/Person.hpp

Insight::Schedule::Event Class Reference

Defines and Object for storing calendar items.
#include <Event.hpp>
Inherits PluginManager::Object.
Inherited by Insight::Schedule::DummyEvent.
Collaboration diagram for Insight::Schedule::Event:
**Detailed Description**

Defines and Object for storing calendar items.

The documentation for this class was generated from the following files:
- Event.hpp
- Event.cpp

---

**Insight::MySQL::Transactions::Exec Class Reference**

Executes the provided SQL code.

```cpp
#include <Exec.hpp>
```

Inherits **Insight::MySQL::SQLTransaction**.

Collaboration diagram for Insight::MySQL::Transactions::Exec:
Detailed Description
Executes the provided SQL code.

The documentation for this class was generated from the following files:
- Exec.hpp
- Exec.cpp

DAL::Field< T > Class Template Reference

#include <Field.hpp>
Inherits DAL::FieldBase.
Collaboration diagram for DAL::Field< T >:
Public Member Functions

- virtual const T & operator* () const
- virtual const T * operator& () const
- virtual std::string ()
- virtual std::string () const
- FieldBase & operator=(const std::string &value)
- template<typename U > FieldBase & operator= (const U &u)

Detailed Description

template<typename T> class DAL::Field< T >

This class wraps a primitive such that its name can be stored along with it's value. Note: there is no non-const direct access to the underlying variable as such access could result in modification that would not set the Altered flag.
Member Function Documentation

template<typename T> virtual DAL::Field< T >::operator std::string () const [inline, virtual]

Returns:
a std::string representation of the underlying value
Implements DAL::FieldBase (p.119).

template<typename T> virtual DAL::Field< T >::operator std::string () [inline, virtual] // Copy

Returns:
a std::string representation of the underlying value
Implements DAL::FieldBase (p.119).

template<typename T> virtual const T* DAL::Field< T >::operator& () const [inline, virtual] // Copy

Returns:
a const pointer to the Field’s underlying value

template<typename T> virtual const T& DAL::Field< T >::operator* () const [inline, virtual] // Copy

Returns:
a const reference to the Field’s underlying value

template<typename T> template<typename U> FieldBase& DAL::Field< T >::operator= (const U & u) [inline]

Assignment operator

Parameters:

\$u\$ the value to be assigned to the Field

Returns:

*this

template<typename T> FieldBase& DAL::Field< T >::operator= (const std::string & value) [inline, virtual] // Copy

Assignment operator

Parameters:

\$value\$ the value to be assigned to the Field

Returns:

*this

Implements DAL::FieldBase (p.120).

The documentation for this class was generated from the following file:

- Field.hpp
DAL::FieldBase Class Reference

#include <FieldBase.hpp>

Inherited by DAL::Field＜T＞, DAL::Lookup, DAL::Reference, DAL::Field＜bool＞, DAL::Field＜boost::posix_time::ptime＞, DAL::Field＜double＞, DAL::Field＜int＞, and DAL::Field＜std::string＞.

Public Member Functions
- FieldBase (const std::string &name) Constructor.
- FieldBase (const FieldBase &field) Copy Constructor.
- virtual const bool & IsAltered () const
- virtual const std::string & GetName () const
- virtual const bool & IsNull () const
- virtual void SetAltered (const bool &altered)
- virtual void SetNull (const bool &null)
- virtual operator std::string ()=0
- virtual operator std::string () const =0
- virtual FieldBase & operator= (const std::string &value)=0

Detailed Description
Abstract base class with common functions for all field types

Constructor & Destructor Documentation

DAL::FieldBase::FieldBase (const std::string & name) [explicit]

Constructor.

Parameters:
  name The label (i.e. column name) by which the field is identified

DAL::FieldBase::FieldBase (const FieldBase & field)

Copy Constructor.

Parameters:
  field
Member Function Documentation

const std::string & DAL::FieldBase::GetName () const  [virtual]

Returns:
The Name (column) of the field

const bool & DAL::FieldBase::IsAltered () const  [virtual]
Indicates whether or not this Field has been altered since it's last interaction with the database.
Returns:
true if the Field has been altered, false otherwise

const bool & DAL::FieldBase::IsNull () const  [virtual]
Indicates whether or not this Field has a value
Returns:
true if NULL, false otherwise

Referenced by Insight::InternetMessageFormat::Message::GenerateMessageId(), Insight::InternetMessageFormat::Message::GeneratePrimaryKeyId(), Insight::RSS::Item::GeneratePrimaryKeyId(), Insight::Directory::Person::ToMailbox(), and Insight::InternetMessageFormat::Message::ToNativeFormat().

Here is the caller graph for this function:

```
virtual DAL::FieldBase::operator std::string () const  [pure virtual]

Returns:
This field's value as represented by a std::string
Implemented in DAL::Field< T > (p.117), DAL::Lookup (p.157), DAL::Reference (p.224), DAL::Field< double > (p.117), DAL::Field< std::string > (p.117), DAL::Field< int > (p.117), DAL::Field< boost::posix_time::ptime > (p.117), and DAL::Field< bool > (p.117).

virtual DAL::FieldBase::operator std::string ()  [pure virtual]

Returns:
This field's value as represented by a std::string
Implemented in DAL::Field< T > (p.117), DAL::Lookup (p.157), DAL::Reference (p.224), DAL::Field< double > (p.117), DAL::Field< std::string > (p.117), DAL::Field< int > (p.117), DAL::Field< boost::posix_time::ptime > (p.117), and DAL::Field< bool > (p.117).```
virtual FieldBase& DAL::FieldBase::operator= (const std::string & value) [pure virtual]

Parameters:

value Converts value to a matching type and sets this Field equal to it.

Returns:

A Reference to this

Implemented in Insight::Communication::Protocol::SecurityTypeLookup (p.246), DAL::Field< T >(p.117), DAL::Lookup (p.158), DAL::Reference (p.225), DAL::Field< double >(p.117), DAL::Field< std::string >(p.117), DAL::Field< int >(p.117), DAL::Field< boost::posix_time::ptime >(p.117), and DAL::Field< bool >(p.117).

void DAL::FieldBase::SetAltered (const bool & altered) [virtual]

Sets the Altered flag as specified

Parameters:

altered The new value of the Altered flag

void DAL::FieldBase::SetNull (const bool & null) [virtual]

Sets the Null flag as specified

Parameters:

null The new value of the Null flag

Reimplemented in DAL::Lookup (p.159), and DAL::Reference (p.227).

The documentation for this class was generated from the following files:

- FieldBase.hpp
- FieldBase.cpp

---

**Insight::Frontend Class Reference**

Adds signals to PluginManager::Frontend.

#include <Frontend.hpp>

Inherits PluginManager::Frontend.

Inherited by Insight::GTKmm::Frontend.

Collaboration diagram for Insight::Frontend:
Public Member Functions

- **Frontend** (PluginManager::PlatformServices &services)
- virtual ~Frontend ()
  
  Destructor.

Public Attributes

- boost::signals2::signal< void()> signal_send_all
  emitted when the user does something indicating a desire to send all messages
- boost::signals2::signal< void()> signal_receive_all
  emitted when the user does something indicating a desire to receive messages from all accounts
- boost::signals2::signal< void()> signal_update_accounts
  emitted when the user does something indicating a desire to send and receive in all accounts
- boost::signals2::signal< void(const std::string &primaryKey)> signal_update_account
  emitted when the user does something indicating a desire to send or receive messages for a specific account

Detailed Description

Adds signals to PluginManager::Frontend.

Constructor & Destructor Documentation

**Insight::Frontend::Frontend** (PluginManager::PlatformServices & services)

Constructor

**Parameters:**

services

Reimplemented from PluginManager::Frontend (p.125).

**Insight::Frontend::~Frontend** () [virtual]

Destructor.

Reimplemented from PluginManager::Frontend (p.126).

Member Data Documentation

boost::signals2::signal< void()> Insight::Frontend::signal_receive_all

emitted when the user does something indicating a desire to receive messages from all accounts

Referenced by Insight::Platform::Init().
boost::signals2::signal<void ()> Insight::Frontend::signal_send_all

emitted when the user does something indicating a desire to send all messages
Referenced by Insight::Platform::Init().

boost::signals2::signal<void (const std::string& primaryKey)>
Insight::Frontend::signal_update_account

emitted when the user does something indicating a desire to send or receive messages for a specific account
Referenced by Insight::Platform::Init().

boost::signals2::signal<void ()> Insight::Frontend::signal_update_accounts

emitted when the user does something indicating a desire to send and receive in all accounts
Referenced by Insight::Platform::Init().

The documentation for this class was generated from the following files:
- src/Frontend.hpp
- src/Frontend.cpp

PluginManager::Frontend Class Reference

This has handles external interactions with the system.
#include <Frontend.hpp>
Inherited by Insight::Frontend.
Collaboration diagram for PluginManager::Frontend:
Public Member Functions

- **Frontend** (PlatformServices & services)
  
  Constructor.
- virtual ~Frontend ()
  
  Destructor.

Protected Member Functions

- virtual void handle_user_alert (const std::string &message) const = 0
  
  Displays a message to the user.
- virtual bool handle_user_abort_offer (const std::string &message) const = 0
  
  Asks the user a yes or no question
- virtual bool handle_user_confirm_request (const std::string &message) const = 0
  
  Asks the user an OK or Cancel question.
- virtual std::string handle_user_text_request (const std::string &message, bool hideText) const = 0
  
  Asks the user for a textual response.

Detailed Description

This has handles external interactions with the system.

Constructor & Destructor Documentation

PluginManager::Frontend::Frontend (PlatformServices & services)

Constructor.

Reimplemented in Insight::Frontend (p. 122).

References handle_user_abort_offer(), handle_user_alert(), handle_user_confirm_request(), PluginManager::PlatformServices::signal_user_abort_offer,
PluginManager::PlatformServices::signal_user_alert,
PluginManager::PlatformServices::signal_user_confirm_request, and PluginManager::PlatformServices::signal_user_text_request.

Here is the call graph for this function:
Destructor.
Reimplemented in **Insight::Frontend** (p.122).

### Member Function Documentation

**virtual bool PluginManager::Frontend::handle_user_abort_offer (const std::string & message)** const [protected, pure virtual]

asks the user a yes or no question

Connects to **PlatformServices::signal_user_abort_offer**

**Parameters:**

* message

**Returns:**

The user's response

Referenced by Frontend().

Here is the caller graph for this function:

```
    PluginManager::Frontend::handle_user_abort_offer
    PluginManager::Frontend::Frontend
```

**virtual void PluginManager::Frontend::handle_user_alert (const std::string & message)** const [protected, pure virtual]

Displays a message to the user.

Connects to **PlatformServices::signal_user_abort_offer**

**Parameters:**

* message

Referenced by Frontend().

Here is the caller graph for this function:

```
    PluginManager::Frontend::handle_user_alert
    PluginManager::Frontend::Frontend
```

**virtual bool PluginManager::Frontend::handle_user_confirm_request (const std::string & message)** const [protected, pure virtual]

Asks the user an OK or Cancel question.

Connects to **PlatformServices::signal_user_confirm_request**

**Parameters:**

* message
virtual std::string PluginManager::Frontend::handle_user_text_request (const std::string & message, bool hideText) const [protected, pure virtual]

Asks the user for a textual response.

Connects to PlatformServices::signal_user_text_request

Parameters:

message The question to present to the user
hideText if true, the user's response will be obfuscated

Returns:
The user's response

Referenced by Frontend().

Here is the caller graph for this function:

The documentation for this class was generated from the following files:
- pluginmanager/src/IniParameters.hpp
- pluginmanager/src/IniParameters.cpp

PluginManager::IniParameters Class Reference

Interacts with the INI file.
#include <IniParameters.hpp>

Public Member Functions

- IniParameters ()
  Constructor. Loads Parameters from insight.ini.
- ~IniParameters ()
  Ensures commit is called before program termination.
- const std::string & GetParameter (const std::string &parameterName, const std::string &parameterSection="general")
  Retrieves the specified Parameter from the specified section.
- void SetParameter (const std::string &parameterName, const std::string &parameterValue, const std::string &parameterSection="general")
Sets the specified Parameter in the specified section to the specified value.

- void Commit()
  Writes Parameters to insight.ini.

Static Public Member Functions

- static IniParameters & GetInstance()
  Ensures that this class is a singleton.
- static std::list<std::string> SplitString(const std::string &string)
  Splits the content of a string into a list of strings delimited by semicolons.

Protected Attributes

- ParameterSectionMap Parameters

Detailed Description

Interacts with the INI file.

Deprecated:

Constructor & Destructor Documentation

PluginManager::IniParameters::IniParameters()

Constructor. Loads Parameters from insight.ini.
References Parameters.

PluginManager::IniParameters::~IniParameters()

Ensures commit is called before program termination.
References Commit().

Here is the call graph for this function:

Member Function Documentation

void PluginManager::IniParameters::Commit()

Writes Parameters to insight.ini.
References ~IniParameters().

Here is the caller graph for this function:
IniParameters & PluginManager::IniParameters::GetInstance () [static]

Ensures that this class is a singleton.

Returns:
A reference to the application's IniParameters

const std::string & PluginManager::IniParameters::GetParameter (const std::string & parameterName, const std::string & parameterSection = "general")

Retrieves the specified Parameter from the specified section.

Parameters:
  parameterName
  parameterSection

Exceptions:
  std::invalid_argument if parameterName is not found
  std::runtime_error if parameterSection is not found

Returns:
  A string containing the desired parameter

References Parameters.

void PluginManager::IniParameters::SetParameter (const std::string & parameterName, const std::string & parameterValue, const std::string & parameterSection = "general")

Sets the specified Parameter in the specified section to the specified value.

Parameters:
  parameterName
  parameterValue
  parameterSection

References Parameters.

std::list< std::string > PluginManager::IniParameters::SplitString (const std::string & string) [static]

Splits the content of a string into a list of strings delimited by semicolons.

Parameters:
  string the std::string to split
Returns:
a list of strings containing all the data in string

Member Data Documentation

ParameterSectionMap PluginManager::IniParameters::Parameters [protected]
   Section, Name, Value
   Referenced by GetParameter(), IniParameters(), and SetParameter().

The documentation for this class was generated from the following files:
- IniParameters.hpp
- IniParameters.cpp

Insight::MySQL::Transactions::Insert Class Reference

Converts the provided Object into an SQL insert script.
#include <Insert.hpp>
Inherits Insight::MySQL::SQLTransaction.
Collaboration diagram for Insight::MySQL::Transactions::Insert:
Detailed Description
Converting the provided object into an SQL insert script.

The documentation for this class was generated from the following files:
- Insert.hpp
- Insert.cpp

Insight::Utility::Session::InvalidResponse Class Reference

Thrown by `GetResponse()` if the server response cannot be interpreted.
#include <Session.hpp>
Inherits std::runtime_error.

Detailed Description
Thrown by `GetResponse()` if the server response cannot be interpreted.

Todo:
convert to boost::exception

The documentation for this class was generated from the following file:
- insight_plugin_pop3/Session.hpp

Insight::Utility::Session::InvalidState Class Reference

Thrown by any POP3 command function that is called when the server is in a state that does not allow said function.
#include <Session.hpp>
Inherits std::runtime_error.

Detailed Description
Thrown by any POP3 command function that is called when the server is in a state that does not allow said function.

Todo:
convert to boost::exception

The documentation for this class was generated from the following file:
- insight_plugin_pop3/Session.hpp
Defines an **RSS** message.

```cpp
#include <Item.hpp>
```

Inherits **Insight::Communication::Message**.

Collaboration diagram for `Insight::RSS::Item`:
Public Member Functions

- virtual void GeneratePrimaryKey()
  Sets the Item's primary key to the base64 encoded representation of its AccountID concatenated with its GUID.

- Item(PluginManager::PlatformServices &services)
  Constructor.

- virtual const std::string ToNativeFormat() const

Static Public Member Functions

- static PluginManager::Object * Factory(PluginManager::PlatformServices &services)
  Factory function. Allocates a new Item.

Detailed Description
Defines an RSS message.

Constructor & Destructor Documentation

Insight::RSS::Item::Item(PluginManager::PlatformServices & services)

Constructor.

Parameters:
  services
  References DAL::Class::Add().
  Referenced by Factory().

Here is the call graph for this function:

![Call Graph]

Here is the caller graph for this function:

![Caller Graph]

Member Function Documentation

PluginManager::Object * Insight::RSS::Item::Factory(PluginManager::PlatformServices & services) [static]

Factory function. Allocates a new Item.
Parameters:

services

Returns:

a pointer to the new Item

References Item().

Here is the call graph for this function:

```
Insight::RSS::Item::Factory ➔ Insight::RSS::Item::Item ➔ DAL::Class::Add
```

void Insight::RSS::Item::GeneratePrimaryKey () [virtual]

Sets the Item's primary key to the base64 encoded representation of its AccountID concatenated with its GUID.

Reimplemented from PluginManager::Object (p.174).

References Insight::Communication::Message::AccountID, DAL::FieldBase::IsNull(), and PluginManager::Object::NewKeyAtInsert.

Here is the call graph for this function:

```
Insight::RSS::Item::GeneratePrimaryKey ➔ DAL::FieldBase::IsNull
```

const std::string Insight::RSS::Item::ToNativeFormat () const [virtual]

Formats a message as it would appear when downloaded.

Implements Insight::Communication::Message (p.170).

The documentation for this class was generated from the following files:

- Item.hpp
- Item.cpp

Framework::Librarian Class Reference

provides the facilites for loading and unloading dynamic libraries

```cpp
#include <Librarian.hpp>
```

Inherited by Framework::Registry.

Collaboration diagram for Framework::Librarian:
Public Member Functions

- **Librarian ()**
  Constructor. Adds options to OptionsDescriptions.

- **virtual ~Librarian ()**
  Calls Unload().

- **virtual void Init (int argc, char *argv[])**
  Scans for libraries and loads them into memory.

- **virtual boost::program_options::options_description & GetOptionDescriptions ()**

- **virtual boost::program_options::variables_map & GetOptions ()**

Protected Member Functions

- **virtual void Scan ()**

- **virtual void Load ()**

- **virtual void Load (const std::string &name)**

- **virtual void Unload ()**

- **virtual void Unload (const std::string &name)**

---

Detailed Description

provides the facilities for loading and unloading dynamic libraries

---

Constructor & Destructor Documentation

Framework::Librarian::Librarian ()

Constructor. Adds options to OptionsDescriptions.

**Returns:**

Framework::Librarian::~Librarian () [virtual]

Calls Unload().

References Unload().

Here is the call graph for this function:

![Call Graph Diagram]
Member Function Documentation

**boost::program_options::options_description & Framework::Librarian::GetOptionDescriptions () [virtual]**

**Returns:**
A reference to the available program options

Referenced by Framework::Managers::BackendManager::BackendManager().

Here is the caller graph for this function:

```
Framework::Librarian::GetOptionDescriptions -> Framework::Managers::BackendManager::BackendManager
```

**boost::program_options::variables_map & Framework::Librarian::GetOptions () [virtual]**

**Returns:**
A reference to the configured program options

Referenced by Framework::Managers::BackendManager::Get().

Here is the caller graph for this function:

```
Framework::Librarian::GetOptions -> Framework::Managers::BackendManager::Get -> Framework::Managers::BackendManager::BackendManager
```

**void Framework::Librarian::Init (int argc, char * argv[]) [virtual]**

scans for libraries and loads them into memory.

**Parameters:**

*argc*  
*argv*

Reimplemented in Framework::Platform (p.197), and Framework::Registry (p.240).

References Load(), and Scan().

Here is the call graph for this function:

```
Framework::Librarian::Init -> Framework::Librarian::Load -> Framework::Librarian::Scan
```

**void Framework::Librarian::Load (const std::string & name) [protected, virtual]**

Allocates a new Library instance and adds it to Libraries

**Parameters:**

*name*  
The name of the Library to load

References PluginManager::PlatformServices::LogEvent, and PluginManager::PlatformServices::signal_library_loaded.
void Framework::Librarian::Load () [protected, virtual]
Loads all of the Libraries in LibraryPaths

Postcondition:
Libraries and LibraryInfo have been populated
Referenced by Init(), and Framework::Registry::Register().

Here is the caller graph for this function:

void Framework::Librarian::Scan () [protected, virtual]
Scans the directories specified by the INI file and attempts to register any Library file found there

Postcondition:
LibraryPaths has been populated
Referenced by Init().

Here is the caller graph for this function:

void Framework::Librarian::Unload (const std::string & name) [protected, virtual]
Calls the Library's unload function and removes it from Libraries

Parameters:
name The name of the Library to unload
References PluginManager::PlatformServices::LogEvent, PluginManager::PlatformServices::signal_library_unloaded.

void Framework::Librarian::Unload () [protected, virtual]
Unloads all Libraries
Referenced by Framework::Registry::Register(), and ~Librarian().

Here is the caller graph for this function:

The documentation for this class was generated from the following files:

- Librarian.hpp
- Librarian.cpp
PluginManager::Library Class Reference

The **Library** class provides a low-level interface for working with dynamic libraries.

```cpp
#include <Library.hpp>
```

Inherits boost::noncopyable.

### Public Member Functions
- **Library** (const std::string &path)
  
  Constructor. Loads the specified library.

- **~Library** ()
  
  Frees the library.

- **void * GetSymbol** (const std::string &symbolName)
  
  Retrieves a specified symbol from the loaded library.

### Protected Member Functions
- **Library** ()
  
  Sets Handle to NULL; This constructor should never be called.

---

**Detailed Description**

The **Library** class provides a low-level interface for working with dynamic libraries.

Since Windows and Posix systems interact with dynamic libraries in different ways, preprocessor directives are used to implement the appropriate platform-specific code.

---

**Constructor & Destructor Documentation**

**PluginManager::Library::Library** (const std::string & path)

Constructs a **Library** object. Loads the specified library.

**Parameters:**

- `path` the path of the library to load

**PluginManager::Library::~Library** ()

Frees the library.

**PluginManager::Library::Library** () [protected]

Sets Handle to NULL; This constructor should never be called.
Member Function Documentation

void * PluginManager::Library::GetSymbol (const std::string & symbolName)

Retrieves a specified symbol from the loaded library.

Exceptions:
std::runtime_error if Handle equals NULL.

Parameters:
symbolName the function or variable name to retrieve from the library

Returns:
NULL if the symbolName is not found, otherwise, returns the symbol

The documentation for this class was generated from the following files:
- Library.hpp
- Library.cpp

Framework::Registry::LibraryAlreadyActivated Struct Reference

Exception.
#include <Registry.hpp>
Inherits std::exception, and boost::exception.

Detailed Description
Exception.

The documentation for this struct was generated from the following file:
- Registry.hpp

PluginManager::LibraryData Struct Reference

Contains Data about a Library.
#include <LibraryData.hpp>
Collaboration diagram for PluginManager::LibraryData:
Public Attributes

- `std::string LibraryName
  The name of the Library. Should correspond to the file name.`

- `Version LibraryVersion
  The version of the Library. This is primarily just for information.`

- `Version PluginAPIVersion
  The version of the PluginAPI for which the Library was written. Used for compatibility checking.`

- `std::list< std::pair< std::string, std::string > > ProvidedObjects
  Lists the Object supertypes and types that are provided by the Library.`

- `std::list< std::string > ProvidedListeners
  Lists the Listeners provided by the library.`

- `std::list< std::string > ProvidedSerializers
  Lists the Serializers provided by the library.`

- `std::list< std::string > ProvidedBackends
  Lists the Backends provided by the Library.`

- `std::list< std::string > RequiredLibraries
  Lists the other Libraries on which the Library depends.`

Detailed Description

Contains Data about a Library.
Member Data Documentation

std::string PluginManager::LibraryData::LibraryName

The name of the Library. Should correspond to the file name.
Referenced by PluginManager::register_plugin().

Version PluginManager::LibraryData::LibraryVersion

The version of the Library. This is primarily just for information.
Referenced by PluginManager::register_plugin().

Version PluginManager::LibraryData::PluginAPIVersion

The version of the PluginAPI for which the Library was written. Used for compatibility checking.
Referenced by PluginManager::register_plugin().

std::list<std::string> PluginManager::LibraryData::ProvidedBackends

Lists the Backends provided by the Library.

std::list<std::string> PluginManager::LibraryData::ProvidedListeners

Lists the Listeners provided by the library.
Referenced by PluginManager::register_plugin().

std::list<std::pair<std::string, std::string>> PluginManager::LibraryData::ProvidedObjects

Lists the Object supertypes and types that are provided by the Library.
Referenced by PluginManager::register_plugin().

std::list<std::string> PluginManager::LibraryData::ProvidedSerializers

Lists the Serializers provided by the library.

std::list<std::string> PluginManager::LibraryData::RequiredLibraries

Lists the other Libraries on which the Library depends.

The documentation for this struct was generated from the following file:

- LibraryData.hpp
Framework::Registry::LibraryNameMismatch Struct Reference

Exception.
#include <Registry.hpp>
Inherits std::exception, and boost::exception.

Detailed Description
Exception.

The documentation for this struct was generated from the following file:
- Registry.hpp

Framework::Registry::LibraryNotRegistered Struct Reference

Exception.
#include <Registry.hpp>
Inherits std::exception, and boost::exception.

Detailed Description
Exception.

The documentation for this struct was generated from the following file:
- Registry.hpp

PluginManager::Listener Class Reference

Generic class from which signal handling classes should be derived.
#include <Listener.hpp>
Inherited by Framework::Interface::Synchronizer, and Insight::Communication::Parser.
Collaboration diagram for PluginManager::Listener:
Public Member Functions

• **Listener (PlatformServices & services)**
  Constructor.

• virtual **~Listener ()**
  Destructor.

Public Attributes

• boost::mutex **Mutex**
  Mutex. Used to lock access to the *Object*.

Detailed Description

Generic class from which signal handling classes should be derived.

Constructor & Destructor Documentation

**PluginManager::Listener::Listener (PlatformServices & services)**

Constructor.

**Parameters:**
- *services* reference to the program's *PlatformServices* structure

**virtual PluginManager::Listener::~Listener ()** [inline, virtual]

Destructor.

Member Data Documentation

**boost::mutex PluginManager::Listener::Mutex**

Mutex. Used to lock access to the *Object*.

The documentation for this class was generated from the following files:

- Listener.hpp
- Listener.cpp
Framework::Managers::ListenerManager Class Reference

Manages the available listeners.
#include <ListenerManager.hpp>
Collaboration diagram for Framework::Managers::ListenerManager:
Public Member Functions

- **ListenerManager** (Registry &registry, PluginManager::PlatformServices &services, PluginManager::RegistrationServices &regServices)
  Constructor. Binds functions and adds options.
- virtual ~ListenerManager ()
- PluginManager::Listener & Get (const std::string &name)
  Retrieves the specified Listener.

Protected Member Functions

- virtual void Activate (boost::shared_ptr< PluginManager::Listener > listener, const std::string &name)

Detailed Description

Manages the available listeners.

Constructor & Destructor Documentation

Framework::Managers::ListenerManager::ListenerManager (Registry & registry, PluginManager::PlatformServices & services, PluginManager::RegistrationServices & regServices)

Constructor. Binds functions and adds options.

**Parameters:**
- registry
- services
- regServices

References PluginManager::RegistrationServices::ActivateListener, PluginManager::RegistrationServices::DeactivateListener, PluginManager::Listener::Get(), and PluginManager::PlatformServices::GetListener.

Here is the call graph for this function:

```
Framework::Managers::ListenerManager::ListenerManager
```

Framework::Managers::ListenerManager::~ListenerManager () [virtual]

Member Function Documentation

```cpp
void Framework::Managers::ListenerManager::Activate (boost::shared_ptr<
    PluginManager::Listener > listener, const std::string & name) [protected, virtual]

    Used by RegistrationServices

    Parameters:
    
    listener
    name

PluginManager::Listener & Framework::Managers::ListenerManager::Get (const std::string & name)

    Retrieves the specified Listener.

    Parameters:
    
    name the name of the Listener to retrieve

    Returns:
    
    A reference to the specified Listener

    Referenced by ListenerManager().
```

Here is the caller graph for this function:

```
Framework::Managers::ListenerManager::Get   Framework::Managers::ListenerManager::ListenerManager
```

The documentation for this class was generated from the following files:
- ListenerManager.hpp
- ListenerManager.cpp

Framework::Logger Class Reference

Provides facilities for logging events.

```cpp
#include <Logger.hpp>
```

Collaboration diagram for Framework::Logger:
Public Member Functions

- **Logger** (PluginManager::PlatformServices &services)
  Constructor. Sets default MinimumPriorityLevel.

- **virtual void LogEvent** (const std::string &module, const PluginManager::PlatformServices::Priority_t &level, const int &lineNumber, const std::string &function, const std::string &message)
  logs an event

- **virtual void Init** (int argc, char *argv[])
  retrieves the minimum log level and which modules to log from the backend

Detailed Description

Provides facilities for logging events.

Constructor & Destructor Documentation

**Framework::Logger::Logger** (PluginManager::PlatformServices & services)

Constructor. Sets default MinimumPriorityLevel.

**Parameters:**

services

References LogEvent(), and PluginManager::PlatformServices::LogEvent.

Here is the call graph for this function:

- Framework::Logger::Logger
- Framework::Logger::LogEvent

Member Function Documentation

**void Framework::Logger::Init** (int argc, char * argv[]) [virtual]

retrieves the minimum log level and which modules to log from the backend

**Parameters:**

argc
argv

References PluginManager::PlatformServices::GetPreference, PluginManager::PlatformServices::Query, and PluginManager::PlatformServices::SetPreference.
void Framework::Logger::LogEvent (const std::string & module,   const PluginManager::PlatformServices::Priority_t & level,   const int & lineNumber,   const std::string & function,   const std::string & message) [virtual]

logs an event

**Parameters:**

- **module** Effectively, this is the plugin in which the event occurred
- **level** severity of the event
- **lineNumber** Line number on which the event occurred
- **function** Function in which the event occurred
- **message** the message to log

Referenced by Logger().

Here is the caller graph for this function:

```
Framework::Logger::LogEvent       Framework::Logger::Logger
```

The documentation for this class was generated from the following files:

- Logger.hpp
- Logger.cpp

---

**DAL::Lookup Class Reference**

The **Lookup** class represents a blend between an enum and a **DAL::Reference**. It has fixed values like an enum, but stores them in a table.

```
#include <Lookup.hpp>
```

Inherits **DAL::FieldBase**.

Inherited by **Insight::Communication::Protocol::SecurityTypeLookup**.

Collaboration diagram for DAL::Lookup:

---
Public Member Functions

- **Lookup** (const std::string &name)
  Constructor. Sets Field's label.

- **Lookup** (const Lookup &field)
  Copy Constructor.

- virtual const std::map<unsigned int, std::string> & **GetPossibleValues** () const
  returns the valid values for this instance

- virtual **operator std::string** ()
Conversion operator.
- virtual operator std::string () const
  Conversion operator.
- virtual operator unsigned int ()
  Conversion operator.
- virtual operator unsigned int () const
  Conversion operator.
- virtual Lookup & operator=(const std::string &value)
  Assignment operator.
- virtual void SetNull (const bool &null)
- virtual bool operator==(const std::string &other) const
- virtual bool operator!=(const std::string &other) const
- virtual bool operator==(const Lookup &other) const
- virtual bool operator!=(const Lookup &other) const

Protected Member Functions

- virtual void AddValue (const std::string &value)
  adds a possible value to this instance. Should only be used in the constructor.

Protected Attributes

- unsigned int Value
  The values of this field.
- std::map< unsigned int, std::string > Index
  Maps integral values to textual values.
- std::map< std::string, unsigned int > InvertedIndex
  Maps textual values to integral values.

Detailed Description

The Lookup class represents a blend between an enum and a DAL::Reference. It has fixed values like an enum, but stores them in a table.

Constructor & Destructor Documentation

DAL::Lookup::Lookup (const std::string & name) [explicit]

Constructor. Sets Field's label.

Parameters:
  name the label of this Field

DAL::Lookup::Lookup (const Lookup & field)

Copy Constructor.
Parameters:

- `field`

---

Member Function Documentation

`void DAL::Lookup::AddValue (const std::string & value) [protected, virtual]`

adds a possible value to this instance. Should only be used in the constructor

Parameters:

- `value`
  References Index, and InvertedIndex.

`const std::map< unsigned int, std::string > & DAL::Lookup::GetPossibleValues () const [virtual]`

returns the valid values for this instance

Returns:

- a map relating the instance's integral values with their correspond semantic labels
  References Index.

`DAL::Lookup::operator std::string () const [virtual]`

Conversion operator.

Returns:

- represents a string representation of the class's value.
  Implements `DAL::FieldBase (p. 119)`. References Index, and Value.

`DAL::Lookup::operator std::string () [virtual]`

Conversion operator.

Returns:

- represents a string representation of the class's value.
  Implements `DAL::FieldBase (p. 119)`. References Index, and Value.
DAL::Lookup::operator unsigned int () const [virtual]

Conversion operator.

**Returns:**
represents an integral representation of the class's value.
References Value.

DAL::Lookup::operator unsigned int () [virtual]

Conversion operator.

**Returns:**
represents an integral representation of the class's value.
References Value.

bool DAL::Lookup::operator!= (const Lookup & other) const [virtual]

**Parameters:**
other

**Returns:**

bool DAL::Lookup::operator!= (const std::string & other) const [virtual]

**Parameters:**
other

**Returns:**

Lookup & DAL::Lookup::operator= (const std::string & value) [virtual]

Assignment operator.

**Parameters:**
value

**Returns:**

Implements DAL::FieldBase (p.120).
Reimplemented in Insight::Communication::Protocol::SecurityTypeLookup (p.246).
References Index, InvertedIndex, SetNull(), and Value.
Here is the call graph for this function:
bool DAL::Lookup::operator== (const Lookup & other) const [virtual]

Parameters:
    other

Returns:

References Index, and Value.

bool DAL::Lookup::operator== (const std::string & other) const [virtual]

Parameters:
    other

Returns:

References Index, and Value.

void DAL::Lookup::SetNull (const bool & null) [virtual]

Sets the Null flag as specified

Parameters:
    null The new value of the Null flag

Reimplemented from DAL::FieldBase (p.120).

References Value.

Referenced by operator=().

Here is the caller graph for this function:

Member Data Documentation

std::map<unsigned int, std::string> DAL::Lookup::Index [protected]

Maps integral values to textual values.

Referenced by AddValue(), GetPossibleValues(), operator std::string(), operator=(), and operator==().

std::map<std::string, unsigned int> DAL::Lookup::InvertedIndex [protected]

Maps textual values to integral values.

Referenced by AddValue(), and operator=().
unsigned int DAL::Lookup::Value [protected]

The values of this field.
Referenced by operator std::string(), operator unsigned int(), operator=(), operator==(), and SetNull().

The documentation for this class was generated from the following files:
- Lookup.hpp
- Lookup.cpp

DAL::Collections::Map< KEY_TYPE, VAL_TYPE > Class Template Reference

Wraps std::map<KEY_TYPE, VAL_TYPE> in a DAL::Collection.
#include <Map.hpp>
Inherits STL::Collection< std::map< KEY_TYPE, VAL_TYPE >, std::pair< const KEY_TYPE, VAL_TYPE >, const std::pair< const KEY_TYPE, VAL_TYPE > >.
Collaboration diagram for DAL::Collections::Map< KEY_TYPE, VAL_TYPE >:
Public Member Functions

- virtual void SetAltered(const bool &altered)
  
  Can be used to explicitly override the altered state (mostly used by Transactions after they have done their work).

- virtual bool IsAltered() const
  
  Indicates if any of Collections fields have been changed since the last database interaction.

Protected Member Functions

- virtual void InsertDummy()
  
  Inserts a dummy entry for schema generation.

---

Detailed Description

template<typename KEY_TYPE, typename VAL_TYPE> class DAL::Collections::Map<KEY_TYPE, VAL_TYPE>

Wraps std::map<KEY_TYPE, VAL_TYPE> in a DAL::Collection.

---

Member Function Documentation

template<typename KEY_TYPE, typename VAL_TYPE> virtual void DAL::Collections::Map<KEY_TYPE, VAL_TYPE>::InsertDummy() [inline, protected, virtual]

Inserts a dummy entry for schema generation.

Implements DAL::Collection (p.93).

template<typename KEY_TYPE, typename VAL_TYPE> virtual bool DAL::Collections::Map<KEY_TYPE, VAL_TYPE>::IsAltered() const [inline, virtual]

Indicates if any of Collections fields have been changed since the last database interaction.

Returns:

true if an alteration has occurred, false otherwise

Reimplemented from DAL::STLCollection< std::map<KEY_TYPE, VAL_TYPE>, std::pair<const KEY_TYPE, VAL_TYPE>, const std::pair<const KEY_TYPE, VAL_TYPE>> (p.299).

Referenced by DAL::Collections::Map<std::string, std::string>::IsAltered().

Here is the caller graph for this function:
template<typename KEY_TYPE, typename VAL_TYPE> virtual void DAL::Collections::Map<KEY_TYPE, VAL_TYPE>::SetAltered (const bool & altered) [inline, virtual]

Can be used to explicitly override the altered state (mostly used by Transactions after they have done their work.

Parameters:

altered The new value for Altered

Reimplemented from DAL::STLCollection< std::map< KEY_TYPE, VAL_TYPE >, std::pair< const KEY_TYPE, VAL_TYPE >, const std::pair< const KEY_TYPE, VAL_TYPE > > (p.300).

Referenced by DAL::Collections::Map< std::string, std::string >::SetAltered().

Here is the caller graph for this function:

![Caller Graph]

The documentation for this class was generated from the following file:

- Map.hpp

---

**Insight::InternetMessageFormat::Message Class Reference**

Represents an RFC 2821 formatted Internet Message.

```
#include <Message.hpp>
```

Inherits **Insight::Communication::Message**.

Collaboration diagram for Insight::InternetMessageFormat::Message:
Public Types

- enum RecipientType
  
  *Used by AddRecipient to determine which field to add to.*

Public Member Functions

- virtual const std::string ToNativeFormat () const
- virtual void GenerateMessageId ()
  
  *Generates a message ID according to the requirements specified by RFC 2822.*
- virtual void GeneratePrimaryKey ()
- virtual void AddRecipient (Directory::Contact &contact, const RecipientType &type)
  
  *Adds a Contact to the appropriate field*
- virtual void SetField (const std::string &name, const std::string &value)
  
  *Sets the value of an optional field.*

Static Public Member Functions

- static PluginManager::Object * Factory (PluginManager::PlatformServices &services)

Protected Member Functions

- virtual void handle_primaryKey_changed (const std::string &oldKey, const std::string &newKey)

Friends

- class Insight::InternetMessageFormat::Parser

Detailed Description

*Represents an RFC 2821 formatted Internet Message.*

Member Enumeration Documentation

enum Insight::InternetMessageFormat::Message::RecipientType

*Used by AddRecipient to determine which field to add to.*

Member Function Documentation

void Insight::InternetMessageFormat::Message::AddRecipient (Directory::Contact & contact, const RecipientType & type) [virtual]

*Adds a Contact to the appropriate field*
Parameters:

- contact The Contact to add
- type the field to which to add the Contact

References DAL::Collections::ReferenceSequence::push_back().

Here is the call graph for this function:

```
Insight::InternetMessageFormat::Message::AddRecipient → DAL::Collections::ReferenceSequence::push_back
```

PluginManager::Object * Insight::InternetMessageFormat::Message::Factory
(PluginManager::PlatformServices & services) [static]

Creates a new InternetMessageFormat

Parameters:

- services

Returns:

A pointer to the new Message

Referenced by PluginManager::activate_plugin().

Here is the caller graph for this function:

```
Insight::InternetMessageFormat::Message::Factory ← PluginManager::activate_plugin
```

void Insight::InternetMessageFormat::Message::GenerateMessageId () [virtual]

Generates a message ID according to the requirements specified by RFC 2822.

The algorithm generates a message ID of the form <unique_text@domain_name>. unique_text is a base64 encoded representation of the date and time that the message is sent.

References Insight::Communication::Message::AccountID, Insight::Communication::Account::GetHost(), DAL::Class::GetPrimaryKey(), and DAL::FieldBase::IsNull().

Referenced by GeneratePrimaryKey().

Here is the call graph for this function:

```
Insight::InternetMessageFormat::Message::GenerateMessageId ← Insight::Communication::Account::GetHost
```

Here is the caller graph for this function:

```
Insight::InternetMessageFormat::Message::GenerateMessageId ← Insight::InternetMessageFormat::Message::GeneratePrimaryKey
```

void Insight::InternetMessageFormat::Message::GeneratePrimaryKey () [virtual]

Generates a primary key based on a combination AccountID MessageId if it exists or DateSent if it does not.

Reimplemented from PluginManager::Object (p.174).
void Insight::InternetMessageFormat::Message::handle_primaryKey_changed (const std::string & oldKey, const std::string & newKey) [protected, virtual]

Updates AccountID if necessary and calls parent version

Parameters:
  oldKey
  newKey

Reimplemented from Insight::Communication::Message (p.170).

Here is the call graph for this function:

void Insight::InternetMessageFormat::Message::SetField (const std::string & name, const std::string & value) [virtual]

Sets the value of an optional field.

Parameters:
  name The name of the field
  value The value to give the field

References DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::end().

Here is the call graph for this function:

const std::string Insight::InternetMessageFormat::Message::ToNativeFormat () const [virtual]

Formats a message as it would appear when downloaded.

Implements Insight::Communication::Message (p.170).

References DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::begin(), DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::empty(), DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::end(), and DAL::FieldBase::IsNull().

Here is the call graph for this function:
The documentation for this class was generated from the following files:

- plugin_internetmessageformat/Message.hpp
- plugin_internetmessageformat/Message.cpp

**Insight::Communication::Message Class Reference**

Defines an interface from which communication messages (email, rss, etc) can be derived.

```cpp
#include <Message.hpp>
```

Inherits **PluginManager::Object**.

Inherited by **Insight::Communication::DummyMessage**, **Insight::InternetMessageFormat::Message**, and **Insight::RSS::Item**.

Collaboration diagram for **Insight::Communication::Message**:
Public Member Functions
- virtual const std::string ToNativeFormat() const = 0

Protected Member Functions
- virtual void handle_primaryKey_changed(const std::string &oldKey, const std::string &newKey)

Protected Attributes
- DAL::Reference AccountID

Detailed Description
Defines an interface from which communication messages (email, rss, etc) can be derived.

Member Function Documentation

void Insight::Communication::Message::handle_primaryKey_changed(const std::string &oldKey, const std::string &newKey) [protected, virtual]

Updates AccountID if necessary and calls parent version

Parameters:
oldKey
newKey

Reimplemented from PluginManager::Object (p.175).
Reimplemented in Insight::InternetMessageFormat::Message (p.167).
References AccountID, and DAL::Reference::Rekey().

Here is the call graph for this function:

virtual const std::string Insight::Communication::Message::ToNativeFormat() const [pure virtual]

Formats a message as it would appear when downloaded.

Implemented in Insight::Communication::DummyMessage (p.109),
Insight::InternetMessageFormat::Message (p.167), and Insight::RSS::Item (p.136).

Member Data Documentation

DAL::Reference Insight::Communication::Message::AccountID [protected]

ObjectIdentifier of the account through which this message was downloaded.

Referenced by Insight::InternetMessageFormat::Message::GenerateMessageId(),
Insight::InternetMessageFormat::Message::GeneratePrimaryKey(), Insight::RSS::Item::GeneratePrimaryKey(),
and handle_primaryKey_changed().
Insight::Utility::Session::NegativeResponse Class Reference

Thrown by `GetResponse()` if the server response begins with "-ERR".

```
#include <Session.hpp>
```

Inherits `std::runtime_error`.

**Detailed Description**

Thrown by `GetResponse()` if the server response begins with "-ERR".

**Todo:**

- convert to `boost::exception`

PluginManager::Object Class Reference

Extends `DAL` class to include common values like modification and creation dates.

```
#include <Object.hpp>
```

Inherits `boost::noncopyable`, and `DAL::Class`.

Inherited by `Insight::Communication::Account`, `Insight::Communication::Message`, `Insight::Directory::Contact`, `Insight::Schedule::Event`, and `PluginManager::DummyObject`.

Collaboration diagram for `PluginManager::Object`:
Public Member Functions

- virtual ~Object ()
  Destructor. Disconnects connected signals.
- const DAL::Field<boost::posix_time::ptime> & GetCreationDate () const
- const DAL::Field<boost::posix_time::ptime> & GetModificationDate () const
- void SetModificationDate (const boost::posix_time::ptime timestamp)
  sets the Object's modification date
- const DAL::Field<std::string> & GetSupertype () const
- virtual const bool & GetNewKeyAtInsert () const
  Indicates whether or not the Object should generate a new primary key when it is saved.
- virtual void SetNewKeyAtInsert (const bool &value)
  enables or disables the regeneration of the Object's primary key when it is saved.
- virtual void GeneratePrimaryKey ()
  Generates a primary key for the Object.
- virtual void Rekey (const std::string &newKey)
  Sets a new PrimaryKey and emits PlatformServices::signal_primaryKey_changed.
- virtual void SetKey (const std::string &newKey)
  THIS FUNCTION FOR INTERNAL USE ONLY. Plugins should use Rekey instead.
- virtual const DAL::Field<std::string> & GetUserID () const

Public Attributes

- boost::shared_mutex Mutex

Protected Member Functions

- virtual void handle_primaryKey_changed (const std::string &oldKey, const std::string &newKey)

Protected Attributes

- DAL::Field<boost::posix_time::ptime> CreationDate
- DAL::Field<std::string> Type
- DAL::Field<std::string> Supertype
- DAL::Field<std::string> UserID
- bool NewKeyAtInsert

Detailed Description

Extends DAL class to include common values like modification and creation dates.

Constructor & Destructor Documentation

PluginManager::Object::~Object () [virtual]
Destructor. Disconnects connected signals.

---

### Member Function Documentation

**void PluginManager::Object::GeneratePrimaryKey () [virtual]**

Generates a primary key for the `Object`.
This calls `Class::GeneratePrimaryKey`, but adds the additional procedure of firing `PlatformServices::signal primaryKey_changed`
Reimplemented from `DAL::Class` *(p.88)*.
Reimplemented in `Insight::Directory::Contact` *(p.99)*, `Insight::InternetMessageFormat::Message` *(p.166)*, `Insight::RSS::Account` *(p.70)*, and `Insight::RSS::Item` *(p.136)*.
References `PluginManager::PlatformServices::signal primaryKey_changed`.

**const DAL::Field< boost::posix_time::ptime > & PluginManager::Object::GetCreationDate () const**

**Returns:**
- The Object's creation date
- References `CreationDate`.

**const DAL::Field< boost::posix_time::ptime > & PluginManager::Object::GetModificationDate () const**

**Returns:**
- The Object's modification date
- Referenced by `Insight::HostSynchronizer::Peer::handle_object_updated()`.

Here is the caller graph for this function:

```
PluginManager::Object::GetModificationDate       Insight::HostSynchronizer::Peer::handle_object_updated
```

**const bool & PluginManager::Object::GetNewKeyAtInsert () const [virtual]**

Indicates whether or no the `Object` should generator a new primary key when it is saved.

**Returns:**
- true if a new primary key should be generated
- References `NewKeyAtInsert`.

**const DAL::Field< std::string > & PluginManager::Object::GetSupertype () const**

**Returns:**
- The name of the lowest class in libinsight from which the instance is derived.
References Supertype.

```cpp
const DAL::Field< std::string > & PluginManager::Object::GetUserID () const [virtual]
```

**Returns:**
The user id with which an Object is associated
References UserID.

```cpp
void PluginManager::Object::handle_primaryKey_changed (const std::string & oldKey, const std::string & newKey) [protected, virtual]
```

Updates the primary keys of any of the collections that belong to this instance

**Parameters:**
- `oldKey`
- `newKey`

Reimplemented in Insight::Communication::Message (p.170), Insight::Directory::Person (p.194), and Insight::InternetMessageFormat::Message (p.167).

```cpp
void PluginManager::Object::Rekey (const std::string & newKey) [virtual]
```

Sets a new PrimaryKey and emits PlatformServices::signal_primaryKey_changed.

**Parameters:**
- `newKey` the new key to assign

References PluginManager::PlatformServices::signal_primaryKey_changed.

```cpp
void PluginManager::Object::SetKey (const std::string & newKey) [virtual]
```

THIS FUNCTION FOR INTERNAL USE ONLY. Plugins should use Rekey instead.
assigns a new key to the Object but _does not_ emit signal_primaryKey_changed.

**Parameters:**
- `newKey` the new key to assign

References SetNewKeyAtInsert().

Here is the call graph for this function:

```
PluginManager::Object::SetKey PluginManager::Object::SetNewKeyAtInsert
```

```cpp
void PluginManager::Object::SetModificationDate (const boost::posix_time::ptime & timestamp)
```

sets the Object's modification date

**Parameters:**
- `timestamp` the modification date
void PluginManager::Object::SetNewKeyAtInsert (const bool & value)  [virtual]

enables or disables the regeneration of the Object's primary key when it is saved.

Parameters:
value if true, then the next time this Object is saved, it will receive a new key
References NewKeyAtInsert.
Referenced by SetKey().
Here is the caller graph for this function:

void PluginManager::Object::SetUserID (const std::string & userId)  [virtual]

sets the user id. Used by SocialEmailSynchronizer

Parameters:
userId
References UserID.

Member Data Documentation

DAL::Field<boost::posix_time::ptime> PluginManager::Object::CreationDate  [protected]
The date this Object was created
Referenced by Insight::InternetMessageFormat::Message::GeneratePrimaryKey(), and GetCreationDate().

boost::shared_mutex PluginManager::Object::Mutex
Mutex

bool PluginManager::Object::NewKeyAtInsert  [protected]
If this value is set to true, GeneratePrimaryKey should be called by the Backend before doing an insert.
Referenced by Insight::InternetMessageFormat::Message::GeneratePrimaryKey(),
Insight::RSS::Item::GeneratePrimaryKey(),
Insight::Directory::Contact::GeneratePrimaryKey(),
GetNewKeyAtInsert(), and SetNewKeyAtInsert().

DAL::Field<std::string> PluginManager::Object::Supertype  [protected]
The name of the lowest non-plugin class from which the instance is derived.
Referenced by GetSupertype().

DAL::Field<std::string> PluginManager::Object::Type  [protected]
Corresponds to table name.
**DAL::Field<std::string> PluginManager::Object::UserID** [protected]

Represents the user to whom this **Object** corresponds. Ideally, this should be a Reference to a contact and it should be added in libinsight, but time is short.

Referenced by GetUserID(), and SetUserID().

The documentation for this class was generated from the following files:

- Object.hpp
- Object.cpp

---

**Framework::Managers::ObjectManager Class Reference**

Keeps track of allocated Objects and provides functions for retrieving and saving them.

```cpp
#include <ObjectManager.hpp>
```

Collaboration diagram for Framework::Managers::ObjectManager:
Public Member Functions

- **ObjectManager** *(Registry &registry, PluginManager::PlatformServices &services, PluginManager::RegistrationServices &regServices)*
  Constructor. Binds functions and adds Options.

- virtual ~**ObjectManager** ()
  Destructor. Deallocates Objects and Factories.

- virtual **PluginManager::Object & Get** (const std::string &primaryKey)
  Retrieves a reference to the specified Object.

- virtual **PluginManager::Object & Make** (const std::string &supertype, const std::string &type)
  Generates a new Object of the specified supertype and type.

- virtual **PluginManager::Object & MakeWithKey** (const std::string &supertype, const std::string &type, const std::string &primaryKey)

- virtual void **Put** (const std::string &primaryKey)
  Saves the specified Object.

- virtual bool **IsOpen** (const std::string &primaryKey)
  Determines whether the specified Object is in memory.

- virtual void **Delete** (const std::string &primaryKey)
  Deletes the specified Object.

- virtual void **Activate** (const std::string &supertype, const std::string &type)
  Activates the Plugin that contains the specified Object types.

- virtual void **Activate** (PluginManager::fn_object_factory factory, const std::string &supertype, const std::string &type)
  Used by RegistrationServices register ObjectFactories.

- virtual void **DeactivateFactory** (const std::string &supertype, const std::string &type)
  deactivates the factory for the specified Object type

---

Detailed Description

Keeps track of allocated Objects and provides functions for retrieving and saving them.

---

Constructor & Destructor Documentation

**Framework::Managers::ObjectManager::ObjectManager** *(Registry & registry, PluginManager::PlatformServices & services, PluginManager::RegistrationServices & regServices)*

Constructor. Binds functions and adds Options.

**Parameters:**

- registry
- services
- regServices
Returns:

References Activate(), PluginManager::RegistrationServices::ActivateObjectFactory, DeactivateFactory(), PluginManager::RegistrationServices::DeactivateObjectFactory, Delete(), PluginManager::PlatformServices::DeleteObject, Get(), PluginManager::PlatformServices::GetObject, isOpen(), PluginManager::PlatformServices::IsOpen, Make(), PluginManager::PlatformServices::MakeKeyedObject, PluginManager::PlatformServices::MakeObject, MakeWithKey(), Put(), PluginManager::PlatformServices::PutObject, PluginManager::PlatformServices::signal_object_closed, PluginManager::PlatformServices::signal_object_deleted, and PluginManager::PlatformServices::signal_primaryKey_changed.

Here is the call graph for this function:

Framework::Managers::ObjectManager::~ObjectManager () [virtual]

Destructor. Deallocates Objects and Factories.

Returns:

Member Function Documentation

void Framework::Managers::ObjectManager::Activate (PluginManager::fn_object_factory factory, const std::string & supertype, const std::string & type) [virtual]

Used by RegistrationServices register ObjectFactories.

Parameters:

factory
supertype
type

void Framework::Managers::ObjectManager::Activate (const std::string & supertype, const std::string & type) [virtual]

Activates the Plugin that contains the specified Object types.
Parameters:

* supertype
* type

References Framework::Registry::Activate(), and Framework::Registry::ListObjectLibraries().

Referenced by Make(), MakeWithKey(), and ObjectManager().

Here is the call graph for this function:

```
void Framework::Managers::ObjectManager::DeactivateFactory (const std::string & supertype, const std::string & type) [virtual]
```

deactivates the factory for the specified Object type

Parameters:

* supertype
* type

Referenced by ObjectManager().

Here is the caller graph for this function:

```
void Framework::Managers::ObjectManager::Delete (const std::string & primaryKey) [virtual]
```

Deletes the specified Object.

Parameters:

* primaryKey

References PluginManager::PlatformServices::GetBackend.

Referenced by ObjectManager().

Here is the caller graph for this function:
PluginManager::Object & Framework::Managers::ObjectManager::Get (const std::string & primaryKey) [virtual]

Retrieves a reference to the specified Object.
If necessary, the Object is retrieved from the Backend.

**Parameters:**
- primaryKey

**Returns:**

References PluginManager::PlatformServices::GetBackend.
Referenced by ObjectManager().

Here is the caller graph for this function:

```
Framework::Managers::ObjectManager::Get.dispatch
```

bool Framework::Managers::ObjectManager::IsOpen (const std::string & primaryKey) [virtual]

Determines whether the specified Object is in memory.

**Parameters:**
- primaryKey

**Returns:**

Referenced by ObjectManager().

Here is the caller graph for this function:

```
Framework::Managers::ObjectManager::IsOpen.dispatch
```

PluginManager::Object & Framework::Managers::ObjectManager::Make (const std::string & supertype, const std::string & type) [virtual]

Generates a new Object of the specified supertype and type.

**Parameters:**
- supertype
- type

**Returns:**

- a reference to the generated Object
  References Activate().
  Referenced by ObjectManager().

Here is the call graph for this function:

```
Framework::Managers::ObjectManager::Make.dispatch
```
PluginManager::Object & Framework::Managers::ObjectManager::MakeWithKey (const std::string & supertype, const std::string & type, const std::string & primaryKey) [virtual]

Generates a new Object of the specified supertype and type with the specified primary key

**Parameters:**
- `supertype`
- `type`
- `primaryKey`

**Returns:**
a reference to the generated Object

References Activate().

Referenced by ObjectManager().

Here is the call graph for this function:

```
Framework::Managers::ObjectManager::MakeWithKey -> Framework::Managers::ObjectManager::Activate -> Framework::Registry::Activate
```

Here is the caller graph for this function:

```
Framework::Managers::ObjectManager::MakeWithKey
```

void Framework::Managers::ObjectManager::Put (const std::string & primaryKey) [virtual]

Saves the specified Object.

**Parameters:**
- `primaryKey`

References PluginManager::PlatformServices::GetBackend.

Referenced by ObjectManager().

Here is the caller graph for this function:

```
Framework::Managers::ObjectManager::Put
```

The documentation for this class was generated from the following files:
- ObjectManager.hpp
- ObjectManager.cpp
Insight::InternetMessageFormat::Parser Class Reference

Parses raw Internet Messages into Objects.
#include <Parser.hpp>
Inherits Insight::Communication::Parser.

Public Member Functions
- Parser (PluginManager::PlatformServices &services)
  Constructor.

Static Protected Member Functions
- static const std::string GetField (const std::string fieldName, std::multimap< std::string, std::string > &messageMap)
  returns the first value keyed on fieldName and removes it from the map

Detailed Description
Parses raw Internet Messages into Objects.

Constructor & Destructor Documentation

Insight::InternetMessageFormat::Parser::Parser (PluginManager::PlatformServices & services)

  Constructor.
  
  Parameters:
  
  services

Member Function Documentation

const std::string Insight::InternetMessageFormat::Parser::GetField (const std::string fieldName, std::multimap< std::string, std::string > & messageMap) [static, protected]

  returns the first value keyed on fieldName and removes it from the map
  
  Parameters:
  
  fieldName the name of the field to retrieve
  messageMap the map from which to retrieve the field
Yoohoo::Peer Class Reference

Processes incoming Yoohoo commands.
#include <Peer.hpp>
Inherited by Insight::HostSynchronizer::Peer.

Public Member Functions

- **Peer** (const std::string &hostId, const std::string &host, const unsigned short &port)
  Constructor. Initializes the Worker thread to keep IoService running.

- **virtual ~Peer ()**
  Destructor. Terminates Worker thread.

- **virtual void ConnectToHost** (const std::string &host, const int &port)
  Connects to host on port and stores the Session in TempSessions.

- **virtual boost::asio::io_service & GetIoService ()**

Detailed Description

Processes incoming Yoohoo commands.

Constructor & Destructor Documentation

Yoohoo::Peer::Peer (const std::string & hostId, const std::string & host, const unsigned short & port)

Constructor. Initializes the Worker thread to keep IoService running.

Parameters:

- **hostId** This host's host id
- **host** This host's routable hostname or ip address
- **port** The port on which this host is listening for incoming connections.

Yoohoo::Peer::~Peer () [virtual]

Destructor. Terminates Worker thread.
**Returns:**

Reimplemented in Insight::HostSynchronizer::Peer (p.188).

---

**Member Function Documentation**

```cpp
void Yoohoo::Peer::ConnectToHost (const std::string & host, const int & port) [virtual]
```

Connects to host on port and stores the Session in TempSessions.

**Parameters:**

- `host` the hostname or IP address of the remote host
- `port` the port on which the remote host is accepting connects.

```cpp
boost::asio::io_service & Yoohoo::Peer::GetIoService () [virtual]
```

Returns a reference to this Session's io_service

**Returns:**

---

The documentation for this class was generated from the following files:

- Yoohoo/Peer.hpp
- Yoohoo/Peer.cpp

---

**Insight::HostSynchronizer::Peer Class Reference**

Processes incoming Synchronization commands.

#include <Peer.hpp>

Inherits Yoohoo::Peer, and Framework::Interface::Synchronizer.

Collaboration diagram for Insight::HostSynchronizer::Peer:
Public Member Functions

- virtual ~Peer ()
  Destructor. Terminates Worker thread.
- virtual void Init ()
  Starts Yoohoo connection to known hosts.
- virtual std::list<std::string> GetHostIds () const
- virtual boost::shared_ptr<Session> GetSession (const std::string &hostId)

Static Public Member Functions

- static const std::string GenerateHostId ()

Protected Member Functions

- virtual void handle_object_changed (const std::string &primaryKey)
  Encodes the changed fields of the object specified by primaryKey using a Serializer. The encoded string is placed in StagedChanges.
- virtual void handle_object_deleted (const std::string &primaryKey)
  Relays a deletion to the other members of the Synchronization group.
- virtual void handle_object_updated (const std::string &primaryKey)
  Calls handle_object_committed.
- virtual void handle_object_committed (const std::string &primaryKey)
  Moves a committed encoded Object string from StagedChanges to CommittedChanges to be picked up by DoWork.
- virtual void handle_primaryKey_changed (const std::string &oldId, const std::string &newId)
  Updates the PrimaryKey of an Object in StagedChanges or CommittedChanges. Also, relays the change to the Synchronization group.
- virtual void handle_preference_changed (const std::string &module, const std::string &name, const std::string &value)
  Does nothing at this level, but provides option for derived classes to deal with changed preferences.
- virtual void DoWork ()

Protected Attributes

- KeyMap UpdatedKeys

Detailed Description

Processes incoming Synchronization commands.

Constructor & Destructor Documentation

Insight::HostSynchronizer::Peer::~Peer () [virtual]

Destructor. Terminates Worker thread.
**Member Function Documentation**

**void Insight::HostSynchronizer::Peer::DoWork () [protected, virtual]**

*doesn’t do anything. Makes compiler happy Todo:*  
remove DoWork from parent declaration

Implements Framework::Interface::Synchronizer (p.304).

**const std::string Insight::HostSynchronizer::Peer::GenerateHostId () [static]**

Generates a host ID. Note: if the profile is moved from one machine to another, the host id preference should be erased otherwise BAD THINGS WILL HAPPEN

**Returns:**

References Utility::Base64Encode().

Here is the call graph for this function:

```
Insight::HostSynchronizer::Peer::GenerateHostId --> Utility::Base64Encode
```

**std::list< std::string > Insight::HostSynchronizer::Peer::GetHostIds () const [virtual]**

**Returns:**

a list containing hostIds of the connected Sessions

**boost::shared_ptr< Session > Insight::HostSynchronizer::Peer::GetSession (const std::string & hostId) [virtual]**

Returns the Session connected to the specified host. If no such connection exists, attempts to create one.

**Parameters:**

*hostId* the HostId of the desired session

**Returns:**

a shared_ptr correponding to hostId

Referenced by handle_object_committed(), handle_object_deleted(), and handle_primaryKey_changed().

Here is the caller graph for this function:

```
Insight::HostSynchronizer::Peer::GetSession --> Insight::HostSynchronizer::Peer:handle_object_committed
Insight::HostSynchronizer::Peer::GetSession --> Insight::HostSynchronizer::Peer:handle_object_deleted
Insight::HostSynchronizer::Peer::GetSession --> Insight::HostSynchronizer::Peer:handle_primaryKey_changed
```
void Insight::HostSynchronizer::Peer::handle_object_changed (const std::string & *primaryKey*)
[protected, virtual]

Encodes the changed fields of the object specified by primaryKey using a Serializer. The encoded string is placed in StagedChanges.

**Parameters:**
- `primaryKey` the PrimaryKey of the changed Object
Reimplemented from Framework::Interface::Synchronizer (p.304).

void Insight::HostSynchronizer::Peer::handle_object_committed (const std::string & *primaryKey*)
[protected, virtual]

Moves a committed encoded Object string from StagedChanges to CommitedChanges to be picked up by DoWork.

**Parameters:**
- `primaryKey` the PrimaryKey of the committed Object
Reimplemented from Framework::Interface::Synchronizer (p.305).

ReferencesgetSession().

Here is the call graph for this function:

```
Insight::HostSynchronizer::Peer::handle_object_committed └→ Insight::HostSynchronizer::Peer::GetSession
```

void Insight::HostSynchronizer::Peer::handle_object_deleted (const std::string & *primaryKey*)
[protected, virtual]

relays a deletion to the other members of the Synchronization group

**Parameters:**
- `primaryKey` the PrimaryKey of the deleted Object
Reimplemented from Framework::Interface::Synchronizer (p.305).

ReferencesgetSession().

Here is the call graph for this function:

```
Insight::HostSynchronizer::Peer::handle_object_deleted └→ Insight::HostSynchronizer::Peer::GetSession
```

void Insight::HostSynchronizer::Peer::handle_object_updated (const std::string & *primaryKey*)
[protected, virtual]

Calls handle_object_committed.
**Parameters:**

`primaryKey`

Reimplemented from Framework::Interface::Synchronizer (p.305).

References PluginManager::Object::GetModificationDate().

Here is the call graph for this function:

void Insight::HostSynchronizer::Peer::handle_preference_changed (const std::string & module, const std::string & name, const std::string & value) [protected, virtual]

Does nothing at this level, but provides option for derived classes to deal with changed preferences.

**Parameters:**

`module`
`name`
`value`

Reimplemented from Framework::Interface::Synchronizer (p.306).

void Insight::HostSynchronizer::Peer::handle_primaryKey_changed (const std::string & oldId, const std::string & newId) [protected, virtual]

Updates the PrimaryKey of an Object in StagedChanges or CommittedChanges. Also, relays the change to the Synchronization group.

**Parameters:**

`oldId`
`newId`

Reimplemented from Framework::Interface::Synchronizer (p.306).

References GetSession().

Here is the call graph for this function:

void Insight::HostSynchronizer::Peer::Init () [virtual]

Starts Yoohoo connection to known hosts.

Loads the Hosts table from the backend, attempts a Yoohoo to each host, and finally sends Discover to each reachable host.

Reimplemented from Yoohoo::Peer (p.185).
Member Data Documentation

KeyMap Insight::HostSynchronizer::Peer::UpdatedKeys  [protected]
keeps track of changed primary keys

The documentation for this class was generated from the following files:
- HostSynchronizer/Peer.hpp
- HostSynchronizer/Peer.cpp

Insight::Directory::Person Class Reference

adds content to Contact that makes sense for people but not for companies or organizations
#include <Person.hpp>
Inherits Insight::Directory::Contact.
Inherited by Insight::Directory::DummyPerson.
Collaboration diagram for Insight::Directory::Person:
Public Member Functions

- virtual const std::string ToMailbox () const
  represents a Contact as an RFC 2821 compatible mailbox

Static Public Member Functions

- static PluginManager::Object * Factory (PluginManager::PlatformServices &services)
  Factory function.

- static Person & PersonFromMailbox (PluginManager::PlatformServices &services, const std::string mailbox)
  converts an RFC 2821 compatible mailbox into a Person.

Protected Member Functions

- virtual void handle_primaryKey_changed (const std::string &oldKey, const std::string &newKey)

Detailed Description

adds content to Contact that makes sense for people but not for companies or organizations

Member Function Documentation

PluginManager::Object * Insight::Directory::Person::Factory (PluginManager::PlatformServices & services) [static]

Factory function.

Parameters:
services

>Returns:
a pointer to the newly allocated Person

void Insight::Directory::Person::handle_primaryKey_changed (const std::string & oldKey, const std::string & newKey) [protected, virtual]

Updates Agent and calls parent version

Parameters:
oldKey
newKey

Reimplemented from PluginManager::Object (p.175).
References DAL::Reference::Rekey().

Here is the call graph for this function:
Person & Insight::Directory::Person::PersonFromMailbox (PluginManager::PlatformServices & services, const std::string mailbox) [static]

converts an RFC 2821 compatible mailbox into a Person.
If the Contact is not found in the backend, a new Contact is created.

Parameters:

services
mailbox the Mailbox to convert into a Person

Returns:
a reference to the (possibly newly created) Person
References PluginManager::PlatformServices::GetObject, PluginManager::PlatformServices::MakeObject, and PluginManager::PlatformServices::Query.

const std::string Insight::Directory::Person::ToMailbox () const [virtual]

represents a Contact as an RFC 2821 compatible mailbox

Returns:

Implements Insight::Directory::Contact (p.99).
Reimplemented in Insight::Directory::DummyPerson (p.111).
References DAL::FieldBase::IsNull().
Here is the call graph for this function:

The documentation for this class was generated from the following files:
- src/Directory/Person.hpp
- src/Directory/Person.cpp

Framework::Platform Class Reference

Combines all of the various Managers with Librarian.
#include <Platform.hpp>
Inherits Framework::Registry.
Inherited by Insight::Platform.
Collaboration diagram for Framework::Platform:
Public Member Functions

- **Platform ()**
  Constructor.
- virtual void **Init (int argc, char *argv[])**
- virtual **PluginManager::PlatformServices & GetServices ()**
  retrieves a reference to PlatformServices

Detailed Description

Combines all of the various Managers with Librarian.

Constructor & Destructor Documentation

**Framework::Platform::Platform ()**

Constructor.
Reimplemented in **Insight::Platform (p.200)**.

Member Function Documentation

**PluginManager::PlatformServices & Framework::Platform::GetServices () [virtual]**

retrieves a reference to PlatformServices

**Returns:**

a reference to PlatformServices

**void Framework::Platform::Init (int argc, char * argv[]) [virtual]**


**Parameters:**

- argc
- argv

Reimplemented from **Framework::Registry (p.240)**.

The documentation for this class was generated from the following files:

- pluginmanager/src/Framework/Platform.hpp
Insight::Platform Class Reference

This class is provides the programmatic interface by which User Interfaces should interact with Insight. #include <Platform.hpp>
Inherits Framework::Platform.
Collaboration diagram for Insight::Platform:
Public Member Functions

- **Platform()**
  Constructor.

- **virtual ~Platform()**
  Destructor. Saves and Deallocates accounts.

- **virtual void Init(int argc, char *argv[], Frontend &frontend)**
  Loads Accounts from the backend and binds signals from frontend.

- **virtual void handle_update_account(const std::string &primaryKey)**
  Calls the specified Account's RunThread() function.

- **virtual void handle_update_accounts()**
  calls handle_update_account for every account

- **virtual void handle_send_all()**
  Calls handle_update_account for all sending accounts.

- **virtual void handle_receive_all()**
  Calls handle_update_account for all receiving accounts.

---

Detailed Description

This class is provides the programmatic interface by which User Interfaces should interact with **Insight**.

---

Constructor & Destructor Documentation

**Insight::Platform::Platform()**

Constructor.
Reimplemented from **Framework::Platform (p.197)**.

**Insight::Platform::~Platform() [virtual]**

Destructor. Saves and Deallocates accounts.
References PluginManager::PlatformServices::PutObject.

---

Member Function Documentation

**void Insight::Platform::handle_receive_all() [virtual]**

Calls handle_update_account for all receiving accounts.
Connected to **Frontend::signal_receive_all**
References PluginManager::PlatformServices::GetObject, handle_update_account(), and PluginManager::PlatformServices::Query.
Referenced by Init().

Here is the call graph for this function:

Insight::Platform::handle_send_all \rightarrow Insight::Platform::handle_update_account

Here is the caller graph for this function:

Insight::Platform::handle_send_all \rightarrow Insight::Platform::Init

**void Insight::Platform::handle_send_all () [virtual]**

Calls handle_update_account for all sending accounts.

Connected to `Frontend::signal_send_all`

References `PluginManager::PlatformServices::GetObject`, `handle_update_account()`, and `PluginManager::PlatformServices::Query`.

Referenced by Init().

Here is the call graph for this function:

Insight::Platform::handle_send_all \rightarrow Insight::Platform::handle_update_account

Here is the caller graph for this function:

Insight::Platform::handle_send_all \rightarrow Insight::Platform::Init

**void Insight::Platform::handle_update_account (const std::string & primaryKey) [virtual]**

Calls the specified Account's RunThread() function.

Connected to `Frontend::signal_update_account`

**Parameters:**

`primaryKey` The key of the account to update

References `PluginManager::PlatformServices::GetObject`.

Referenced by handle_receive_all(), handle_send_all(), handle_update_accounts(), and Init().

Here is the caller graph for this function:

Insight::Platform::handle_receive_all

Insight::Platform::handle_update_account

Insight::Platform::handle_send_all

Insight::Platform::Init

Insight::Platform::handle_update_accounts
void Insight::Platform::handle_update_accounts () [virtual]

calls handle_update_account for every account
Connected to Frontend::signal_updates_accounts
References handle_update_account(), and PluginManager::PlatformServices::Query.
Referenced by Init().

Here is the call graph for this function:

```
Insight::Platform::handle_update_accounts   -->   Insight::Platform::handle_update_account
```

Here is the caller graph for this function:

```
Insight::Platform::handle_update_accounts   -->   Insight::Platform::Init
```

void Insight::Platform::Init (int argc,   char * argv[],   Frontend & frontend) [virtual]

Loads Accounts from the backend and binds signals from frontend.

Parameters:

- `argc`
- `argv`
- `frontend`

References PluginManager::PlatformServices::GetObject, handle_receive_all(), handle_send_all(), handle_update_account(), handle_update_accounts(), PluginManager::PlatformServices::Query, Insight::Frontend::signal_receive_all, Insight::Frontend::signal_send_all, Insight::Frontend::signal_update_account, and Insight::Frontend::signal_update_accounts.

Here is the call graph for this function:

```
Insight::Platform::handle_receive_all   -->   Insight::Platform::handle_update_account
Insight::Platform::Init
Insight::Platform::handle_send_all     -->   Insight::Platform::handle_update_accounts
```

The documentation for this class was generated from the following files:

- src/Platform.hpp
- src/Platform.cpp
PluginManager::PlatformServices Struct Reference

This struct contains all of the functions needed to register a plugin and all of the functions needed for plugins to access the capabilities of other plugins.

#include <PlatformServices.hpp>

Collaboration diagram for PluginManager::PlatformServices:
Public Types

- enum Priority_t
  
  provides a means for which to specify how important a logged message is

Public Member Functions

- PlatformServices (int major, int minor)
  
  Initializes the PluginApiVersion struct.

Public Attributes

- Version PluginApiVersion
  
  The current API version.

- boost::function<std::function<Object&(const std::string &supertype, const std::string &type)>> MakeObject
- boost::function<std::function<Object&(const std::string &supertype, const std::string &type, const std::string &primaryKey)>> MakeKeyedObject
- boost::function<std::function<Object&(const std::string &primaryKey)>> GetObject
- boost::function<std::function<void(const std::string &primaryKey)>> PutObject
- boost::function<std::function<void(const std::string &primaryKey)>> DeleteObject
- boost::function<std::function<bool(const std::string &primaryKey)>> IsOpen

  Indicates whether an Object is already open and managed by Librarian.

- boost::function<std::function<Backend&(const std::string &name)>> GetBackend
- boost::function<std::function<Frontend&(const std::string &name)>> GetFrontend
- boost::function<std::function<Listener&(const std::string &name)>> GetListener
- boost::function<std::function<void(const std::string &name)>> InsertLibrary
- boost::function<std::function<void(const std::string &name)>> EjectLibrary
- boost::function<std::function<std::map<std::string, bool>> ListLibraryStatus
  
  Returns a mapping of library names to a flag indicating whether or not they are activated

- boost::function<std::function<const LibraryData&(const std::string &name)>> ListLibraryDetails
  
  Returns a LibraryData struct for the specified library

- boost::signals2::signal<std::function<void(const std::string &message)>> signal_user_alert
  
  Signal for alerting the user to an event

- boost::signals2::signal<std::function<bool(const std::string &message)>> signal_user_abort_offer
  
  Presents an abort (OK/Cancel) dialog to the user

- boost::signals2::signal<std::function<bool(const std::string &message)>> signal_user_confirm_request
  
  Presents a confirmation (yes/no) dialog to the user

- boost::signals2::signal<std::function<void>(const std::string &message, bool hideText)>> signal_user_text_request
  
  Requests text from the user, which can optionally be hidden (e.g. for passwords).

- boost::signals2::signal<std::function<void(const std::string oid)>> signal_object_committed
  
  This signal is emitted when an object is inserted into the Backend.
- boost::signals2::signal<void(const std::string &oid)> signal_object_changed
  This signal is emitted when an object is changed but before it is updated in the backend.

- boost::signals2::signal<void(const std::string &oid)> signal_object_not_changed
  This signal is emitted when an object has been changed, but could not be committed.

- boost::signals2::signal<void(const std::string &oid)> signal_object_updated
  This signal is emitted when an object is updated in the Backend.

- boost::signals2::signal<void(const std::string &oid)> signal_object_deleted
  This signal is emitted when an Object is about to be deleted.

- boost::signals2::signal<void(const std::string &oid)> signal_object_delete_pending
  this signal is emitted when an Object is about to be deleted.

- boost::signals2::signal<void(const std::string &oldId, const std::string &newId)> signal_object_closed
  this signal should be emitted when a function that used GetObject is finished with the Object

- boost::signals2::signal<void(const std::string &pluginName)> signal_library_loaded
  This signal is emitted when a plugin is loaded.

- boost::signals2::signal<void(const std::string &pluginName)> signal_plugin_registered
  This signal is emitted when a plugin is registered.

- boost::signals2::signal<void(const std::string &pluginName)> signal_plugin_activated
  This signal is emitted when a plugin is activated.

- boost::signals2::signal<void(const std::string &pluginName)> signal_library_unloaded
  This signal is emitted when a plugin is unloaded.

- boost::signals2::signal<void(const std::string &pluginName)> signal_plugin_unregistered
  This signal is emitted when a plugin is unregistered.

- boost::signals2::signal<void(const std::string &pluginName)> signal_plugin_deactivated
  This signal is emitted when a plugin is deactivated.
- boost::signals2::signal< void(const std::string &module, const Priority_t &level, const int &lineNumber, const std::string &function, const std::string &message)> LogEvent
  Logs an event.
- boost::function< void(const std::string &module, const std::string &name, const std::string &value)> SetPreference
  sets the value of the specified preference
- boost::function< const std::string(const std::string &module, const std::string &name)> GetPreference
  gets the value of the specified preference
- boost::signals2::signal< void(const std::string &module, const std::string &name, const std::string &value)> signal_preference_changed
  This signal is emitted when the specified preference is changed.
- boost::function< Backend::QueryResult(const std::string &sql)> Query
  This function is the same as calling Services.GetBackend("default").Query(sql).
- boost::function< void(const std::string &sql)> Exec
  This function is the same as calling Services.GetBackend("default").Exec(sql).

---

**Detailed Description**

This struct contains all of the functions needed to register a plugin and all of the functions needed for plugins to access the capabilities of other plugins.

---

**Member Enumeration Documentation**

```cpp
enum PluginManager::PlatformServices::Priority_t
{
    // enumeration values
}
```

provides a means for which to specify how important a logged message is

---

**Constructor & Destructor Documentation**

```cpp
PluginManager::PlatformServices::PlatformServices (int major, int minor) [inline]
```

Initializes the PluginApiVersion struct.

---

**Member Data Documentation**

```cpp
boost::function<void(const std::string& primaryKey)> PluginManager::PlatformServices::DeleteObject
```

Deletes the specified Object.

**Parameters:**

- primaryKey
Referenced by Framework::Managers::ObjectManager::ObjectManager().

```cpp
boost::function<Object& (const std::string& format, const std::string& objectString)>
PluginManager::PlatformServices::Deserialize
```

Deserializes the specified `Object` string using the specified format.
Referenced by Framework::Managers::SerializerManager::SerializerManager().

```cpp
boost::function<Object&(const std::string& format, const std::string& filename)>
PluginManager::PlatformServices::DeserializeFromFile
```

Deserializes the specified `Object` string using the specified format from the specified file.
Referenced by Framework::Managers::SerializerManager::SerializerManager().

```cpp
boost::function<void(const std::string& name)>
PluginManager::PlatformServices::EjectLibrary
```

Unloads the specified `Library`.

**Parameters:**

- `name` the name of the `Library` to unload

```cpp
boost::function<void (const std::string& sql)>
PluginManager::PlatformServices::Exec
```

This function is the same as calling Services.GetBackend("default").Exec(sql).
Referenced by Framework::Registry::Activate().

```cpp
boost::function<Backend& (const std::string& name)>
PluginManager::PlatformServices::GetBackend
```

Retrieves the specified `Backend`.

**Parameters:**

- `name` The name of the `Backend`

**Returns:**

- a reference to the specified `Backend`

Referenced by PluginManager::activate_plugin(), Framework::Managers::BackendManager::BackendManager(), Framework::Managers::ObjectManager::Delete(), Framework::Managers::ObjectManager::Get(), and Framework::Managers::ObjectManager::Put().

```cpp
boost::function<Frontend& (const std::string& name)>
PluginManager::PlatformServices::GetFrontend
```

Retrieves the specified `Frontend`.
Parameters:

name The name of the Frontend

Returns:

a reference to the specified Frontend

boost::function<Listener& (const std::string& name)>
PluginManager::PlatformServices::GetListener

Retrieves the specified Listener.

Parameters:

name The name of the Listener

Returns:

a reference to the specified Listener

Referenced by Framework::Managers::ListenerManager::ListenerManager().

boost::function<Object& (const std::string& primaryKey)>
PluginManager::PlatformServices::GetObject

Parameters:

primaryKey

Returns:

Referenced by Insight::Platform::handle_receive_all(), Insight::Platform::handle_send_all(),
Insight::Platform::handle_update_account(), Insight::Platform::Init(),
Framework::Managers::ObjectManager::ObjectManager(), and
Insight::Directory::Person::PersonFromMailbox().

boost::function<const std::string(const std::string& module, const std::string& name)>
PluginManager::PlatformServices::GetPreference

gets the value of the specified preference

Referenced by Framework::Logger::Init().

boost::function<void(const std::string& name)>
PluginManager::PlatformServices::InsertLibrary

Loads the specified Library.

Parameters:

name the name of the Library to load

boost::function<bool(const std::string& primaryKey)>
PluginManager::PlatformServices::IsOpen

Indicates whether an Object is already open and managed by Librarian.
Referenced by PluginManager::deactivate_plugin(), and Framework::Managers::ObjectManager::ObjectManager().

**boost::function<std::set<std::string>>** PluginManager::PlatformServices::ListBackends

returns the names of the available Backends  
Referenced by Framework::Registry::Registry().

**boost::function<const LibraryData& (const std::string& name)>** PluginManager::PlatformServices::ListLibraryDetails

returns a **LibraryData** struct for the specified library  
Referenced by Framework::Registry::Registry().

**boost::function<std::map<std::string, bool)>** PluginManager::PlatformServices::ListLibraryStatus

returns a mapping of library names to a flag indicating whether or not they are activated  
Referenced by Framework::Registry::Registry().

**boost::function<std::set<std::string>>** PluginManager::PlatformServices::ListListeners

returns the names of the available Listeners  
Referenced by Framework::Registry::Registry().

**boost::function<std::set<std::string>>** PluginManager::PlatformServices::ListObjectSupertypes

returns the names of the available **Object** supertypes  
Referenced by Framework::Registry::Registry().

**boost::function<std::set<std::string> const std::string& supertype)>** PluginManager::PlatformServices::ListObjectTypes

returns the names of the available Objects  
Referenced by Framework::Registry::Registry().

**boost::function<std::set<std::string>>** PluginManager::PlatformServices::ListSerializers

returns the names of the available Serializers  
Referenced by Framework::Registry::Registry().

**boost::signals2::signal<void(const std::string& module, const Priority_t& level, const int& lineNumber, const std::string& function, const std::string& message)>** PluginManager::PlatformServices::LogEvent
Logs an event.
Referenced by Framework::Registry::Activate(), Framework::Registry::Deactivate(),
Insight::Utility::Session::GetResponse(), Framework::Librarian::Load(), Framework::Logger::Logger(),
Insight::HostSynchronizer::Session::Parse(), Framework::Registry::Register(),
Insight::HostSynchronizer::Session::Send(), Insight::Utility::Session::SendRequest(),
Framework::Librarian::Unload(), and Framework::Registry::Unregister().

boost::function<Object& (const std::string& supertype, const std::string& type, const std::string& primaryKey)> PluginManager::PlatformServices::MakeKeyedObject

Parameters:

  type
  supertype
  primaryKey Overrides the primaryKey to which the Object is mapped. Note: this does not override the
  Object's primaryKey

Returns:

  a newly allocated object that has a smart pointer somewhere
Referenced by Framework::Managers::ObjectManager::ObjectManager().

boost::function<Object& (const std::string& supertype, const std::string& type)> PluginManager::PlatformServices::MakeObject

Parameters:

  type
  supertype

Returns:

  a newly allocated object that has a smart pointer somewhere
Referenced by Framework::Managers::ObjectManager::ObjectManager(), and
Insight::Directory::Person::PersonFromMailbox().

Version PluginManager::PlatformServices::PluginApiVersion

The current API version.
Referenced by Framework::Registry::Register().

boost::function<void(const std::string& primaryKey)> PluginManager::PlatformServices::PutObject

Parameters:

  primaryKey
Referenced by Framework::Managers::ObjectManager::ObjectManager(), and Insight::Platform::~Platform().

boost::function<Backend::QueryResult (const std::string& sql)> PluginManager::PlatformServices::Query

This function is the same as calling Services.GetBackend("default").Query(sql).
Referenced by Framework::Registry::Activate(), PluginManager::deactivate_plugin(), Insight::Platform::handle_send_all(), Insight::Platform::handle_update_accounts(), Insight::Platform::Init(), Framework::Logger::Init(), and Insight::Directory::Person::PersonFromMailbox().

**boost:\function<data\function(const std::string& format, const Object& object, const Mode_t& mode)> PluginManager::PlatformServices::Serialize**

Serializes the specified **Object** using the specified format.
Referenced by Framework::Managers::SerializerManager::SerializerManager().

**boost:\function<data\function(void(const std::string& format, const Object& object, const Mode_t& mode, const std::string& filename)> PluginManager::PlatformServices::SerializeToFile**

Serializes the specified **Object** using the specified format and saves it as the specified file.
Referenced by Framework::Managers::SerializerManager::SerializerManager().

**boost:\function<data\function(const std::string& module, const std::string& name, const std::string& value)> PluginManager::PlatformServices::SetPreference**

Sets the value of the specified preference
Referenced by Framework::Logger::Init().

**boost:\signals\2::signal<void(const std::string& pluginName)> PluginManager::PlatformServices::signal_library_loaded**

This signal is emitted when a plugin is loaded.
Referenced by Framework::Librarian::Load().

**boost:\signals\2::signal<void(const std::string& pluginName)> PluginManager::PlatformServices::signal_library_unloaded**

This signal is emitted when a plugin is unloaded.
Referenced by Framework::Librarian::Unload().

**boost:\signals\2::signal<void(const std::string oid)> PluginManager::PlatformServices::signal_object_changed**

This signal is emitted when an object is changed but before it is updated in the backend.

**Parameters:**

\textit{oid} the UID of the object that was updated

**boost:\signals\2::signal<void(const std::string& oig)> PluginManager::PlatformServices::signal_object_closed**
this signal should be emitted when a function that used GetObject is finished with the Object
Referenced by PluginManager::deactivate_plugin(), and Framework::Managers::ObjectManager::ObjectManager().

boost::signals2::signal<void(const std::string oid)>
PluginManager::PlatformServices::signal_object_committed

This signal is emitted when an object is inserted into the Backend.

**Parameters:**

oid the UID of the object that was committed

boost::signals2::signal<void(const std::string oid)>
PluginManager::PlatformServices::signal_object_delete_pending

this signal is emitted when an Object is about to be deleted.

boost::signals2::signal<void(const std::string oid)>
PluginManager::PlatformServices::signal_object_deleted

This signal is emitted when an object is deleted from the Backend.

**Parameters:**

oid the UID of the object that was updated

Referenced by PluginManager::Backend::Backend(), and Framework::Managers::ObjectManager::ObjectManager().

boost::signals2::signal<void(const std::string oid)>
PluginManager::PlatformServices::signal_object_not_changed

This signal is emitted when an object has been changed, but could not be committed.

**Parameters:**

oid the UID of the object that was updated

boost::signals2::signal<void(const std::string oid)>
PluginManager::PlatformServices::signal_object_updated

This signal is emitted when an object is updated in the Backend.

**Parameters:**

oid the UID of the object that was updated
This signal is emitted when a plugin is activated.
Referenced by Framework::Registry::Activate().

This signal is emitted when a plugin is deactivated.
Referenced by Framework::Registry::Deactivate().

This signal is emitted when a plugin is registered.
Referenced by Framework::Registry::Register().

This signal is emitted when a plugin is unregistered.
Referenced by Framework::Registry::Unregister().

This signal is emitted when the specified preference is changed.

This signal is called when an Object's PrimaryKey is changed by Object::SetIdentifier.
Referenced by PluginManager::Backend::Backend(), PluginManager::Object::GeneratePrimaryKey(), Framework::Managers::ObjectManager::ObjectManager(), and PluginManager::Object::Rekey().

presents an abort (OK/Cancel) dialog to the user

Parameters:

- **message** Text that states what the user has requested and

Referenced by PluginManager::Frontend::Frontend().
**boost::signals2::signal<void(const std::string& message)>**

PluginManager::PlatformServices::signal_user_alert

signal for alerting the user to an event

Referenced by PluginManager::Frontend::Frontend().

**boost::signals2::signal<bool(const std::string& message)>**

PluginManager::PlatformServices::signal_user_confirm_request

presents a confirmation (yes/no) dialog to the user

**Parameters:***

- *message* Text that asks the user a yes or no question

Referenced by PluginManager::Frontend::Frontend().

**boost::signals2::signal<std::string(const std::string& message, bool hideText)>**

PluginManager::PlatformServices::signal_user_text_request

Requests text from the user, which can optionally be hidden (e.g. for passwords).

**Returns:**

The text requested from the user

**Parameters:**

- *message* A message explaining what the user should enter
- *hideText* if true, the text should be treated as a password field

Referenced by PluginManager::Frontend::Frontend().

---

The documentation for this struct was generated from the following file:

- PlatformServices.hpp

---

**Insight::GTKmm::Views::PluginDetails Class Reference**

```
#include <PluginDetails.hpp>
Inherits Insight::GTKmm::View<std::string>.
```

---

**Detailed Description**

Note: this class does not take advantage of View<T>’s indexing abilities

The documentation for this class was generated from the following files:
Insight::Communication::Protocol Class Reference

implements a communications path between a client and a server
#include <Protocol.hpp>
Inherited by Insight::Communication::Receiver, and Insight::Communication::Sender.

Classes
• class SecurityTypeLookup

Defines the security options available for a protocol. Public Member Functions
• boost::thread * Run ()
  
  runs the Protocol’s actions on a separate thread.

Public Attributes
• boost::signals2::signal<void()> signal_connected
  
  Emmitted when an account connects to its server.
• boost::signals2::signal<void()> signal_disconnected
  
  Emitted when account disconnects from the server.

Detailed Description

implements a communications path between a client and a server

Member Function Documentation

boost::thread * Insight::Communication::Protocol::Run ()

  runs the Protocol’s actions on a separate thread.

Member Data Documentation

boost::signals2::signal<void()> Insight::Communication::Protocol::signal_connected

  Emmitted when an account connects to its server.

boost::signals2::signal<void()> Insight::Communication::Protocol::signal_disconnected

  Emitted when account disconnects from the server.
Insight::MySQL::Transactions::Query Class Reference

Executes the specified query.
#include <Query.hpp>
Inherits Insight::MySQL::SQLTransaction.
Collaboration diagram for Insight::MySQL::Transactions::Query:
**Detailed Description**

Executes the specified query.

The documentation for this class was generated from the following files:
- Query.hpp
- Query.cpp

---

**Insight::Communication::Receiver Class Reference**

specialization of **Protocol** for protocols that receive messages

```cpp
#include <Receiver.hpp>
```

Inherits **Insight::Communication::Protocol**.

Inherited by **Insight::POP3::Account**, and **Insight::RSS::Account**.

Collaboration diagram for Insight::Communication::Receiver:

```
+ signal_connected
+ signal_disconnected
+ Run()
# RunThread()

Insight::Communication::Receiver
+ signal_message_received
```

**Public Attributes**

- `boost::signals2::signal< void(const std::string message, const std::string receivingAccount)>
  signal_message_received`

---

**Detailed Description**

specialization of **Protocol** for protocols that receive messages
Member Data Documentation

boost::signals2::signal<void (const std::string message, const std::string receivingAccount)>
Insight::Communication::Receiver::signal_message_received

Parameters:

message the raw downloaded message
accountUid the uid of the account which downloaded the message

Referenced by Insight::RSS::Account::Account().

The documentation for this class was generated from the following file:
- Receiver.hpp

DAL::Reference Class Reference

Represents a foreign key relationship in a database.
#include <Reference.hpp>
Inherits DAL::FieldBase.
Collaboration diagram for DAL::Reference:
Public Member Functions

- **Reference** (const std::string &name, const std::string &foreignGroup, const std::string &foreignName, const std::string &foreignKey)
  Constructor. Initializes the reference if the value of it's FOREIGN KEY is known.

- **Reference** (const std::string &name, const std::string &foreignGroup, const std::string &foreignName)
  Constructor. Initializes the reference if the value of it's FOREIGN KEY is known.

- **Reference** (const std::string &name, const Class &c)
  Constructor. Gets the table, column, and value directly from the provided class. Primarily used by Collection constructor.

- **Reference** (const Reference &other)
  Copy Constructor.

- **Reference** ()
  this function should never be called, but is required by the compiler due to ReferenceSequence. Calling this function will result in an assertion failure

- virtual const std::string & GetForeignGroup () const
  Retrieves the name of the foreign table.

- virtual const std::string & GetForeignName () const
  Retrieves the name of the foreign column.

- virtual void Rekey (const std::string &key)
  Used when the foreign class's key changes but the foreign class is still conceptually the same object.

- virtual Class & operator* ()
- virtual const Class & operator* () const
- virtual void SetNull (const bool &null=true)
- virtual operator std::string ()
  Conversion operator.

- virtual operator std::string () const
  Conversion operator.

- bool operator== (Reference &other)
  Comparison operator.

- bool operator!= (Reference &other)
  Comparison operator.

- Reference & operator= (const Reference &other)
  Assignment operator.

- Reference & operator= (Class &value)
  Assignment operator.

- virtual FieldBase & operator= (const std::string &value)
  Assignment operator.

Protected Attributes

- std::string ForeignGroup
  Refers to the table in which the foreign key is stored.

- std::string ForeignName
  Refers to the column in the foreign table.

- std::string ForeignKey
  Refers to the value in the foreign table.
Detailed Description
Represents a foreign key relationship in a database.

Constructor & Destructor Documentation

DAL::Reference::Reference (const std::string & name, const std::string & foreignGroup, const std::string & foreignName, const std::string & foreignKey)

Constructor. Initializes the reference if the value of it's FOREIGN KEY is known.

Parameters:
- name The column name of this Field
- foreignGroup The table to which this Field refers
- foreignName The column to which this Field refers
- foreignKey The value to which this field refers

DAL::Reference::Reference (const std::string & name, const std::string & foreignGroup, const std::string & foreignName)

Constructor. Initializes the reference if the value of it's FOREIGN KEY is known.

Parameters:
- name The column name of this Field
- foreignGroup The table to which this Field refers
- foreignName The column to which this Field refers

DAL::Reference::Reference (const std::string & name, const Class & c)

Constructor. Gets the table, column, and value directly from the provided class. Primarily used by Collection constructor.

Parameters:
- name The column name of this Field
- c The class to which this Field refers

Returns:

DAL::Reference::Reference (const Reference & other)

Copy Constructor.

Parameters:
- other The Field from which to copy
Returns:

References ForeignGroup, ForeignKey, and ForeignName.

**DAL::Reference::Reference () [inline]**

this function should never be called, but is required by the compiler due to ReferenceSequence. Calling this function will result in an assertion failure

Returns:

---

**Member Function Documentation**

**const std::string & DAL::Reference::GetForeignGroup () const [virtual]**

Retrieves the name of the foreign table.

**Returns:**

The foreign table name
References ForeignGroup.

**const std::string & DAL::Reference::GetForeignName () const [virtual]**

Retrieves the name of the foreign column.

**Returns:**

The foreign column name
References ForeignName.

**DAL::Reference::operator std::string () const [virtual]**

Conversion operator.

**Returns:**

Implements **DAL::FieldBase (p.119).**
References ForeignKey.

**DAL::Reference::operator std::string () [virtual]**

Conversion operator.
**Returns:**

Implements `DAL::FieldBase` \((p.119)\).

References `ForeignKey`.

`bool DAL::Reference::operator!= (Reference & other)`

Comparison operator.

**Parameters:**

other

**Returns:**

`const Class & DAL::Reference::operator* () const` \([virtual]\)

**Returns:**

A mutable reference to the referred-to `Class`

`Class & DAL::Reference::operator* ()` \([virtual]\)

**Returns:**

A mutable reference to the referred-to `Class`

`FieldBase & DAL::Reference::operator= (const std::string & value)` \([virtual]\)

Assignment operator.

**Parameters:**

value

**Returns:**

Implements `DAL::FieldBase` \((p.120)\).

References `ForeignKey`, and `SetNull()`.

Here is the call graph for this function:

```
DAL::Reference::operator=  -----------------> DAL::Reference::SetNull
```

`Reference & DAL::Reference::operator= (Class & value)`

Assignment operator.
Parameters:
value

Returns:
*this
References ForeignKey, and DAL::Class::GetPrimaryKey().

Here is the call graph for this function:

Reference & DAL::Reference::operator= (const Reference & other)

Assignment operator.

Parameters:
other

Returns:
*this
References ForeignKey.

bool DAL::Reference::operator== (Reference & other)

Comparison operator.

Parameters:
other

Returns:

References ForeignKey.

void DAL::Reference::Rekey (const std::string & key) [virtual]

Used when the foreign class's key changes but the foreign class is still conceptually the same object.

Parameters:
key
References ForeignKey.

Referenced by Insight::Directory::Person::handle_primaryKey_changed(), Insight::InternetMessageFormat::Message::handle_primaryKey_changed(), Insight::Communication::Message::handle_primaryKey_changed(), and DAL::Collection::Rekey().

Here is the caller graph for this function:
void DAL::Reference::SetNull (const bool & null = true) [virtual]

Sets the Null flag as specified

Parameters:

null The new value of the Null flag
Reimplemented from DAL::FieldBase (p.120).

References ForeignKey.

Referenced by operator=().

Here is the caller graph for this function:

Member Data Documentation

std::string DAL::Reference::ForeignGroup [protected]

Refers to the table in which the foreign key is stored.

Referenced by GetForeignGroup(), and Reference().

std::string DAL::Reference::ForeignKey [protected]

Refers to the value in the foreign table.

Referenced by operator std::string(), operator=(), operator==( ), Reference(), Rekey(), and SetNull().

std::string DAL::Reference::ForeignName [protected]

Refers to the column in the foreign table.

Referenced by GetForeignName(), and Reference().

The documentation for this class was generated from the following files:

- Reference.hpp
DAL::Collections::ReferenceSequence Class Reference

Specialize DAL::Sequence<VAL_TYPE> for VAL_TYPE = DAL::Reference.
#include <ReferenceSequence.hpp>
Inherits Sequence< Reference >.
Collaboration diagram for DAL::Collections::ReferenceSequence:
Public Member Functions

- **ReferenceSequence** (const std::string &group, const Class &c, const std::string &referenceGroup, const std::string &referenceName)
- virtual void **push_front** (const Class &c)
  *Creates a Reference from c inserts it.*
- virtual void **push_back** (const Class &c)
  *Creates a Reference from c inserts it.*

Protected Member Functions

- virtual void **InsertDummy** ()
  *Inserts a dummy entry for schema generation.*

Detailed Description

Specialize DAL::Sequence<VAL_TYPE> for VAL_TYPE = DAL::Reference.

Constructor & Destructor Documentation

DAL::Collections::ReferenceSequence::ReferenceSequence (const std::string & group, const Class & c, const std::string & referenceGroup, const std::string & referenceName)

**Parameters:**
- group
- c
- referenceGroup
- referenceName

**Returns:**

Member Function Documentation

void DAL::Collections::ReferenceSequence::InsertDummy () [protected, virtual]

Inserts a dummy entry for schema generation.
Reimplemented from DAL::Collections::Sequence< Reference > (p.251).

void DAL::Collections::ReferenceSequence::push_back (const Class & c) [virtual]

Creates a Reference from c inserts it.
Parameters:

\( c \)

Referenced by Insight::InternetMessageFormat::Message::AddRecipient().

Here is the caller graph for this function:

```
DAL::Collections::ReferenceSequence::push_back  
Insight::InternetMessageFormat::Message::AddRecipient
```

```cpp
void DAL::Collections::ReferenceSequence::push_front ( const Class & c ) [ virtual ]
```

Creates a Reference from \( c \) inserts it.

Parameters:

\( c \)

The documentation for this class was generated from the following files:

- ReferenceSequence.hpp
- ReferenceSequence.cpp

---

**PluginManager::RegistrationServices Struct Reference**

Provides functions for registering plugins.

```
#include <RegistrationServices.hpp>
```

Collaboration diagram for PluginManager::RegistrationServices:
Public Member Functions

- **RegistrationServices** (int major, int minor)
  Constructor.

Public Attributes

- **Version APIVersion**
  contains the API version of The Insight Prokect
- boost::function< void(boost::shared_ptr< BackendFactory > factory, const std::string &name)> **ActivateBackendFactory**
  Registers a **Backend** factory.
- boost::function< void(boost::shared_ptr< FrontendFactory > factory, const std::string &name)> **ActivateFrontendFactory**
  Registers a **Frontend** factory.
- boost::function< void(boost::shared_ptr< Listener > listener, const std::string &name)> **ActivateListener**
  Registers a **Listener**.
- boost::function< void(boost::shared_ptr< ObjectFactory > factory, const std::string &supertype, const std::string &type)> **ActivateObjectFactory**
  Registers an **Object** factory.
- boost::function< void(boost::shared_ptr< Serializer > serializer, const std::string &name)> **ActivateSerializer**
  Registers a **Serializer**.
- boost::function< void(const std::string &module, const std::string &name, const std::string &type)> RegisterPreference
  Registers a Preference.
- boost::function< void(const std::string &name)> DeactivateBackendFactory
  Unregisters a Backend factory.
- boost::function< void(const std::string &name)> DeactivateFrontendFactory
  Unregisters a Frontend factory.
- boost::function< void(const std::string &name)> DeactivateListener
  Unregisters a Listener.
- boost::function< void(const std::string &supertype, const std::string &type)> DeactivateObjectFactory
  Unregisters an Object factory.
- boost::function< void(const std::string &name)> DeactivateSerializer
  Unregisters a Serializer.

---

**Detailed Description**
Provides functions for registering plugins.

---

**Constructor & Destructor Documentation**

```cpp
PluginManager::RegistrationServices::RegistrationServices (int major, int minor) [inline]
```

Constructor.

**Parameters:**

- major
- minor

---

**Member Data Documentation**

```cpp
boost::function< void(fn_backend_factory factory, const std::string& name)>
PluginManager::RegistrationServices::ActivateBackendFactory
```

Registers a Backend factory.

Referenced by Framework::Managers::BackendManager::BackendManager().

```cpp
boost::function< void(fn_frontend_factory factory, const std::string& name)>
PluginManager::RegistrationServices::ActivateFrontendFactory
```

Registers a Frontend factory.
**boost::function<void(boost::shared_ptr<Listener> listener, const std::string& name)>**

PluginManager::RegistrationServices::ActivateListener

Registers a **Listener**.

Referenced by PluginManager::activate_plugin(), and Framework::Managers::ListenerManager::ListenerManager().

**boost::function<void(fn_object_factory, const std::string& supertype, const std::string& type)>**

PluginManager::RegistrationServices::ActivateObjectFactory

Registers an **Object** factory.

Referenced by PluginManager::activate_plugin(), and Framework::Managers::ObjectManager::ObjectManager().

**boost::function<void(boost::shared_ptr<Serializer> serializer, const std::string& name)>**

PluginManager::RegistrationServices::ActivateSerializer

Registers a **Serializer**.

Referenced by Framework::Managers::SerializerManager::SerializerManager().

**Version PluginManager::RegistrationServices::APIVersion**

contains the API version of The **Insight** Prokect

**boost::function<void(const std::string& name)>**

PluginManager::RegistrationServices::DeactivateBackendFactory

Unregisters a **Backend** factory.

Referenced by Framework::Managers::BackendManager::BackendManager().

**boost::function<void(const std::string& name)>**

PluginManager::RegistrationServices::DeactivateFrontendFactory

Unregisters a **Frontend** factory.

**boost::function<void(const std::string& name)>**

PluginManager::RegistrationServices::DeactivateListener

Unregisters a **Listener**.

Referenced by PluginManager::deactivate_plugin(), and Framework::Managers::ListenerManager::ListenerManager().

**boost::function<void(const std::string& supertype, const std::string& type)>**

PluginManager::RegistrationServices::DeactivateObjectFactory

Unregisters an **Object** factory.
Referenced by Framework::Managers::ObjectManager::ObjectManager().

\texttt{boost::function<void(const std::string& name)> PluginManager::RegistrationServices::DeactivateSerializer}

Unregisters a \texttt{Serializer}.
Referenced by Framework::Managers::SerializerManager::SerializerManager().

\texttt{boost::function<void(const std::string& module, const std::string& name, const std::string& type)> PluginManager::RegistrationServices::RegisterPreference}

Registers a Preference.
Referenced by PluginManager::activate_plugin().

The documentation for this struct was generated from the following file:
- RegistrationServices.hpp

---

\textbf{Framework::Registry Class Reference}

provides facilities for activating and deactivating plugins
\texttt{#include <Registry.hpp>}
Inherits \texttt{Framework::Librarian}.
Inherited by \texttt{Framework::Platform}.
Collaboration diagram for Framework::Registry:
Classes

- struct LibraryAlreadyActivated
- Exception. struct LibraryNameMismatch
- Exception. struct LibraryNotRegistered
- Exception. struct UnmetDependency
- Exception. struct VersionMismatch

Exception. Public Types

- typedef boost::error_info< struct tag_dependency_name, std::string > _dependency_name
  Exception Error Info.
- typedef boost::error_info< struct tag_file_name, std::string > _file_name
  Exception Error Info.
- typedef boost::error_info< struct tag_expected_major, unsigned int > _expected_major
  Exception Error Info.
- typedef boost::error_info< struct tag_expected_minor, unsigned int > _expected_minor
  Exception Error Info.
- typedef boost::error_info< struct tag_real_major, unsigned int > _real_major
  Exception Error Info.
- typedef boost::error_info< struct tag_real_minor, unsigned int > _real_minor
  Exception Error Info.
- typedef std::map< std::string, boost::shared_ptr< PluginManager::LibraryData > > LibraryDataMap
  Maps library names to library information.
- typedef std::map< std::string, std::string > ProviderMap
  Maps provided types to library names.

Public Member Functions

- Registry ()
  Constructor. Binds signals from PlatformServices.
- virtual void Init (int argc, char *argv[])
  Calls Librarian::Init() and then calls Register to load data about plugins.
- virtual const PluginManager::LibraryData & ListLibraryDetails (const std::string &name) const
- virtual const ProviderMap & ListBackendLibraries () const
- virtual const ProviderMap & ListListenerLibraries () const
- virtual const ProviderMap & ListSerializerLibraries () const
- virtual const std::map< std::string, ProviderMap > & ListObjectLibraries () const
- virtual const std::set< std::string > ListBackends () const
- virtual const std::set< std::string > ListListeners () const
- virtual const std::set< std::string > ListSerializers () const
- const std::set< std::string > ListObjectSupertypes () const
- const std::set< std::string > ListObjectTypes (const std::string &supertype) const
- std::map< std::string, bool > ListActiveLibraries () const
- virtual void Activate (const std::string &name)
  Activates the specified library.
- virtual void Deactivate (const std::string &name)
  Deactivates the specified library.
Protected Member Functions

- virtual void Register ()
  registers all loaded libraries
- virtual void Register (const std::string &name)
  Registers the specified library.
- virtual void Unregister ()
  Unregistered each library.
- virtual void Unregister (const std::string &name)
  Unregisters the specified library.

Detailed Description

provides facilities for activating and deactivating plugins

Member Typedef Documentation

typedef boost::error_info<struct tag_dependency_name , std::string >
Framework::Registry::_dependency_name

Exception Error Info.

typedef boost::error_info<struct tag_expected_major , unsigned int >
Framework::Registry::_expected_major

Exception Error Info.

typedef boost::error_info<struct tag_expected_minor , unsigned int >
Framework::Registry::_expected_minor

Exception Error Info.

typedef boost::error_info<struct tag_file_name , std::string >
Framework::Registry::_file_name

Exception Error Info.

typedef boost::error_info<struct tag_real_major , unsigned int >
Framework::Registry::_real_major

Exception Error Info.

typedef boost::error_info<struct tag_real_minor , unsigned int >
Framework::Registry::_real_minor

Exception Error Info.
typedef std::map<std::string, boost::shared_ptr<PluginManager::LibraryData> > Framework::Registry::LibraryDataMap

Maps library names to library information.

typedef std::map<std::string, std::string> Framework::Registry::ProviderMap

Maps provided types to library names.

---

Constructor & Destructor Documentation

**Framework::Registry::Registry ()**

Constructor. Binds signals from PlatformServices.

References `ListActiveLibraries()`, `ListBackends()`, `PluginManager::PlatformServices::ListBackends`, `ListLibraryDetails()`, `PluginManager::PlatformServices::ListLibraryDetails`, `ListLibraryStatus()`, `PluginManager::PlatformServices::ListLibraryStatus`, `ListListeners()`, `PluginManager::PlatformServices::ListListeners`, `ListObjectSupertypes()`, `PluginManager::PlatformServices::ListObjectSupertypes`, `ListObjectTypes()`, `PluginManager::PlatformServices::ListObjectTypes`, `ListSerializers()`, and `PluginManager::PlatformServices::ListSerializers`.

Here is the call graph for this function:
Member Function Documentation

```cpp
void Framework::Registry::Activate (const std::string & name) [virtual]
```

Activates the specified library.

**Parameters:**

- `name` the name of the library to activate

References PluginManager::PlatformServices::Exec, PluginManager::PlatformServices::LogEvent, PluginManager::PlatformServices::Query, and PluginManager::PlatformServices::signal_plugin_activated.

Referenced by Framework::Managers::ObjectManager::Activate().

Here is the caller graph for this function:

![Caller Graph for Activate]

```cpp
void Framework::Registry::Deactivate (const std::string & name) [virtual]
```

Deactivates the specified library.

**Parameters:**

- `name` the library to deactivate

References Utility::ListFind(), PluginManager::PlatformServices::LogEvent, and PluginManager::PlatformServices::signal_plugin_deactivated.

Referenced by Unregister().

Here is the call graph for this function:

![Call Graph for Deactivate]

Here is the caller graph for this function:

![Caller Graph for Deactivate]

```cpp
void Framework::Registry::Init (int argc, char * argv[]) [virtual]
```

Calls Librian::Init() and then calls Register to load data about plugins.

**Parameters:**

- `argc`
- `argv[]`
Reimplemented from \texttt{Framework::Librarian (p.139)}.  
Reimplemented in \texttt{Framework::Platform (p.197)}.  
References Register().

Here is the call graph for this function:

\begin{tikzpicture}
    \node[fill=gray!50] (A) {\texttt{Framework::Registry::Init}}; \node[fill=gray!50] (B) {\texttt{Framework::Registry::Register}}; \draw[->] (A) -- (B);
\end{tikzpicture}

\begin{verbatim}
std::map< std::string, bool > Framework::Registry::ListActiveLibraries () const

Returns:  
A map containing every library and whether or not it is activated  
Referenced by Registry().
\end{verbatim}

Here is the caller graph for this function:

\begin{tikzpicture}
    \node[fill=gray!50] (A) {\texttt{Framework::Registry::ListActiveLibraries}}; \node[fill=gray!50] (B) {\texttt{Framework::Registry::Registry}}; \draw[->] (A) -- (B);
\end{tikzpicture}

\begin{verbatim}
const Registry::ProviderMap & Framework::Registry::ListBackendLibraries () const [virtual]

Returns:  
the names of the libraries containing backends and the backends they provide
\end{verbatim}

\begin{verbatim}
const std::set< std::string > Framework::Registry::ListBackends () const [virtual]

Returns:  
the names of the available backends  
Referenced by Registry().
\end{verbatim}

Here is the caller graph for this function:

\begin{tikzpicture}
    \node[fill=gray!50] (A) {\texttt{Framework::Registry::ListBackends}}; \node[fill=gray!50] (B) {\texttt{Framework::Registry::Registry}}; \draw[->] (A) -- (B);
\end{tikzpicture}

\begin{verbatim}
const PluginManager::LibraryData & Framework::Registry::ListLibraryDetails (const std::string & name) const [virtual]

Parameters:  
\textit{name} The name of the Library of interest
\end{verbatim}

\begin{verbatim}
Returns:  
information about the specified library  
Referenced by Registry().
\end{verbatim}

Here is the caller graph for this function:

\begin{tikzpicture}
    \node[fill=gray!50] (A) {\texttt{Framework::Registry::ListLibraryDetails}}; \node[fill=gray!50] (B) {\texttt{Framework::Registry::Registry}}; \draw[->] (A) -- (B);
\end{tikzpicture}
const Registry::ProviderMap & Framework::Registry::ListListenerLibraries () const [virtual]

Returns:
the names of the libraries containing listeners and the listeners they provide

const std::set< std::string > Framework::Registry::ListListeners () const [virtual]

Returns:
the names of the available frontends
Referenced by Registry().
Here is the caller graph for this function:

\[
\text{Framework::Registry::ListListeners} \rightarrow \text{Framework::Registry::Registry}
\]

const std::map< std::string, Registry::ProviderMap > & Framework::Registry::ListObjectLibraries () const [virtual]

Returns:
the names of the libraries containing Objects and the Objects they provide
Referenced by Framework::Managers::ObjectManager::Activate().
Here is the caller graph for this function:

\[
\text{Framework::Registry::ListObjectLibraries} \rightarrow \text{Framework::Managers::ObjectManager::Activate} \rightarrow \text{Framework::Managers::ObjectManager::ObjectManager}
\]

const std::set< std::string > Framework::Registry::ListObjectSupertypes () const

Returns:
the names of the available supertypes
Referenced by Registry().
Here is the caller graph for this function:

\[
\text{Framework::Registry::ListObjectSupertypes} \rightarrow \text{Framework::Registry::Registry}
\]

const std::set< std::string > Framework::Registry::ListObjectTypes (const std::string & supertype) const

Returns:
the names of the available types for a given supertype

Parameters:
\textit{supertype} the supertype for which to retrieve types
Referenced by Registry().

Here is the caller graph for this function:

```
Framework::Registry::ListObjectTypes ─── Framework::Registry::Registry
```

**const Registry::ProviderMap & Framework::Registry::ListSerializerLibraries () const** [virtual]

**Returns:**
the names of the libraries containing serializers and the serializers they provide

**const std::set< std::string > Framework::Registry::ListSerializers () const** [virtual]

**Returns:**
the names of the available serializers

Referenced by Registry().

Here is the caller graph for this function:

```
Framework::Registry::ListSerializers ─── Framework::Registry::Registry
```

**void Framework::Registry::Register (const std::string & name)** [protected, virtual]

 Registers the specified library.

**Parameters:**

*name* the name of the library to register

References Framework::Librarian::Load(), PluginManager::PlatformServices::LogEvent, PluginManager::PlatformServices::PluginApiVersion, PluginManager::PlatformServices::signal_plugin_registered, and Framework::Librarian::Unload().

Here is the call graph for this function:

```
Framework::Registry::Register ─── Framework::Librarian::Load
Framework::Registry::Register ─── Framework::Librarian::Unload
```

**void Framework::Registry::Register ()** [protected, virtual]

 registers all loaded libraries

Referenced by Ini().

Here is the caller graph for this function:

```
Framework::Registry::Register ─── Framework::Registry::Init
```

Referenced by Registry().

Here is the caller graph for this function:
void Framework::Registry::Unregister (const std::string & name) [protected, virtual]

Unregisters the specified library.

Parameters:
name the library to unregister

References Deactivate(), PluginManager::PlatformServices::LogEvent, and PluginManager::PlatformServices::signal_plugin_unregistered.

Here is the call graph for this function:

void Framework::Registry::Unregister () [protected, virtual]

Unregistered each library.

The documentation for this class was generated from the following files:
- Registry.hpp
- Registry.cpp

---

**Insight::Communication::Protocol::SecurityTypeLookup Class Reference**

Defines the security options available for a protocol.

```cpp
#include <Protocol.hpp>
```

Inherits `DAL::Lookup`.

Collaboration diagram for Insight::Communication::Protocol::SecurityTypeLookup:
Public Member Functions

- virtual Lookup & \texttt{operator=} (const std::string &value)
  
  Assignment operator.

Detailed Description

Defines the security options available for a protocol.
These options are PLAIN, meaning TCP, and SSL, meaning SSL over TCP

Member Function Documentation

\begin{verbatim}
virtual Lookup& Insight::Communication::Protocol::SecurityTypeLookup::operator= (const std::string & value) [inline, virtual]

Assignment operator.

Parameters:
\begin{itemize}
  \item \texttt{value}
\end{itemize}

Returns:

Reimplemented from \texttt{DAL::Lookup (p.158)}.
\end{verbatim}

The documentation for this class was generated from the following file:

- Protocol.hpp

Insight::MySQL::Transactions::Select Class Reference

Creates an SQL select script for the specified Object and then loads the retrieved data into the Object.
\#include <Select.hpp>

Inherits Insight::MySQL::SQLTransaction.

Collaboration diagram for Insight::MySQL::Transactions::Select:
**Detailed Description**

Creates an SQL select script for the specified Object and then loads the retrieved data into the Object.

The documentation for this class was generated from the following files:
- Select.hpp
- Select.cpp

---

**Insight::Communication::Sender Class Reference**

specialization of Protocol for protocols the send messages
#include <Sender.hpp>
Inherits Insight::Communication::Protocol.
Inherited by Insight::SMTP::Account.
Collaboration diagram for Insight::Communication::Sender:

![Collaboration diagram]

**Detailed Description**

specialization of Protocol for protocols the send messages

The documentation for this class was generated from the following file:
- Sender.hpp
DAL::Collections::Sequence< VAL_TYPE > Class Template Reference

Wraps std::list<VAL_TYPE> in a DAL::Collection.
#include <Sequence.hpp>
Inherits DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >.
Collaboration diagram for DAL::Collections::Sequence< VAL_TYPE >:
Public Member Functions

- virtual void SetAltered (const bool &altered)
  
  Can be used to explicitly override the altered state (mostly used by Transactions after they have done their work).

- virtual bool IsAltered () const
  
  Indicates if any of Collections fields have been changed since the last database interaction.

Protected Member Functions

- virtual void InsertDummy ()
  
  Inserts a dummy entry for schema generation.

Detailed Description

template<typename VAL_TYPE> class DAL::Collections::Sequence< VAL_TYPE >

Wraps std::list<VAL_TYPE> in a DAL::Collection.

Member Function Documentation

template<typename VAL_TYPE> virtual void DAL::Collections::Sequence< VAL_TYPE >::InsertDummy () [inline, protected, virtual]

  Inserts a dummy entry for schema generation.
  Implements DAL::Collection (p.93).
  Reimplemented in DAL::Collections::ReferenceSequence (p.230).

template<typename VAL_TYPE> virtual bool DAL::Collections::Sequence< VAL_TYPE >::IsAltered () const [inline, virtual]

  Indicates if any of Collections fields have been changed since the last database interaction.

  Returns:
  
  true if an alteration has occurred, false otherwise
  
  Reimplemented from DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE > (p.299).
  
  Referenced by DAL::Collections::Sequence< Reference >::IsAltered().
  
  Here is the caller graph for this function:

  
  DAL::Collections::Sequence::IsAltered
template<typename VAL_TYPE> virtual void DAL::Collections::Sequence< VAL_TYPE >::SetAltered (const bool & altered) [inline, virtual]

Can be used to explicitly override the altered state (mostly used by Transactions after they have done their work.

Parameters:

altered The new value for Altered

Reimplemented from DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE > (p.300).

Referenced by DAL::Collections::Sequence< Reference >::SetAltered().

Here is the caller graph for this function:

The documentation for this class was generated from the following file:

- Sequence.hpp

---

PluginManager::Serializer Class Reference

Interface for converting an Object to and from a text format.

#include <Serializer.hpp>

Inherits DAL::Transaction.

Inherited by Insight::XML::Serializer.

Collaboration diagram for PluginManager::Serializer:
Public Member Functions

- virtual void SetMode (const Mode_t &mode=CHANGES)
  
  Used specify whether the entire Object should be encode or just its changed fields.

- virtual Object & operator() (const std::string &objectString)=0
  
  Converts a string to an Object.

- virtual std::string operator() (const Object &object)=0
  
  Converts an Object into a string.

Detailed Description

Interface for converting an Object to and from a text format.

Member Function Documentation

virtual std::string PluginManager::Serializer::operator() (const Object & object) [pure virtual]

Converts an Object into a string.

Parameters:

  object the Object to encode

Returns:

  a std::string representation of the Object

Implemented in Insight::XML::Serializer (p.257).

virtual Object& PluginManager::Serializer::operator() (const std::string & objectString) [pure virtual]

Converts a string to an Object.

Parameters:

  objectString The string to deserialize

Returns:

  The deserialized Object

Implemented in Insight::XML::Serializer (p.257).

void PluginManager::Serializer::SetMode (const Mode_t & mode = CHANGES) [virtual]

Used specify whether the entire Object should be encode or just its changed fields.
Parameters:

mode

Referenced by Framework::Managers::SerializerManager::Serialize().

Here is the caller graph for this function:

The documentation for this class was generated from the following files:

- insight_pluginmanager/src/Serializer.hpp
- insight_pluginmanager/src/Serializer.cpp

**Insight::XML::Serializer Class Reference**

Converts Objects to and from XML.

#include <Serializer.hpp>

Inherits **PluginManager::Serializer**.

Collaboration diagram for Insight::XML::Serializer:
Public Member Functions

- virtual PluginManager::Object & operator() (const std::string &objectString)
  Converts a string to an Object.
- virtual std::string operator() (const PluginManager::Object &object)
  Converts an Object into a string.

Protected Member Functions

- virtual std::string Rubberize (const std::string &rawValue) const
  Makes values character safe.

Detailed Description

Converts Objects to and from XML.

Member Function Documentation

std::string Insight::XML::Serializer::operator() (const PluginManager::Object & object) [virtual]

Converts an Object into a string.

Parameters:
  
  object the Object to encode

Returns:
  
  a std::string representation of the Object

Implements PluginManager::Serializer (p.254).

References DAL::Transaction::GetCollections(), and DAL::Transaction::GetFields().

Here is the call graph for this function:

PluginManager::Object & Insight::XML::Serializer::operator() (const std::string & objectString) [virtual]

Converts a string to an Object.

Parameters:
  
  objectString The string to deserialize
Returns:
The deserialized Object

Implements PluginManager::Serializer (p.254).

References DAL::Transaction::GetCollections(), and DAL::Transaction::GetFields().

Here is the call graph for this function:

std::string Insight::XML::Serializer::Rubberize (const std::string & rawValue) const  [protected, virtual]

Makes values character safe.

Parameters:

rawValue the raw string representation of the value

Returns:

the properly escaped string

The documentation for this class was generated from the following files:

- insight_plugin_xml_serializer/src/Serializer.hpp
- insight_plugin_xml_serializer/src/Serializer.cpp

Framework::Managers::SerializerManager Class Reference

Keeps track of Serializers and provides the functions for adding, using, and removing them.

#include <SerializerManager.hpp>

Collaboration diagram for Framework::Managers::SerializerManager:
Public Member Functions

- **SerializerManager** (Registry & registry, PluginManager::PlatformServices & services, PluginManager::RegistrationServices & regServices)
  Constructor. Binds functions.
- virtual ~SerializerManager ()
  Deallocates Serializers.
- virtual PluginManager::Serializer & Get (const std::string & name)
  Retrieves the specified Serializer.
- virtual std::string Serialize (const std::string & format, const PluginManager::Object & object, const PluginManager::Mode_t & mode)
  Converts the Object to a string in the second format in the specified mode.
- virtual void Serialize (const std::string & format, const PluginManager::Object & object, const PluginManager::Mode_t & mode, const std::string & filename)
  Converts the Object to a string in the second format in the specified mode.
- virtual PluginManager::Object & Deserialize (const std::string & format, const PluginManager::Object & object)
  Creates or modifies an Object from encoded text.
- virtual PluginManager::Object & DeserializeFromFile (const std::string & format, const std::string & filename)
  Creates or modifies an Object from an encoded text file.

Protected Member Functions

- void Activate (boost::shared_ptr< PluginManager::Serializer > serializer, const std::string & name)

Detailed Description

Keeps track of Serializers and provides the functions for adding, using, and removing them.

Constructor & Destructor Documentation

Framework::Managers::SerializerManager::SerializerManager (Registry & registry, PluginManager::PlatformServices & services, PluginManager::RegistrationServices & regServices)

Constructor. Binds functions.

Parameters:

  registry
  services
  regServices

Returns:

References PluginManager::RegistrationServices::ActivateSerializer,
PluginManager::RegistrationServices::DeactivateSerializer,
PluginManager::PlatformServices::Deserialize,
PluginManager::PlatformServices::DeserializeFromFile,
PluginManager::PlatformServices::DeserializeFromFile, Serialize(),
PluginManager::PlatformServices::Serialize, and PluginManager::PlatformServices::SerializeToFile.

Here is the call graph for this function:

```
Framework::Managers::SerializerManager::~SerializerManager () [virtual]
```

Deallocates Serializers.

---

Member Function Documentation

```cpp
void Framework::Managers::SerializerManager::Activate (boost::shared_ptr<
  PluginManager::Serializer > serializer, const std::string & name) [protected]
```

Used by RegistrationServices

Parameters:
- `serializer`
- `name`

---

```cpp
PluginManager::Object & Framework::Managers::SerializerManager::Deserialize (const std::string &
  format, const std::string & objectString) [virtual]
```

Creates or modifies an Object from encoded text.

Parameters:
- `format` the format in which the Object is encoded
- `objectString` the encoded-text representation of the Object

Returns:
A reference to the newly allocated Object

Referenced by DeserializeFromFile(), and SerializerManager().

Here is the call graph for this function:

```
Framework::Managers::SerializerManager::Deserialize
```

Here is the caller graph for this function:

```
Framework::Managers::SerializerManager::Deserialize
```

```cpp
PluginManager::Object & Framework::Managers::SerializerManager::DeserializeFromFile (const std::string & format,
  const std::string & filename) [virtual]
```
Creates or modifies an Object from an encoded text file.

**Parameters:**
- `format` the format in which the Object is encoded
- `filename` The name of the file containing the encoded Object

**Returns:**
A reference to the newly allocated Object
References Deserialize().
Referenced by SerializerManager().

Here is the call graph for this function:

Here is the caller graph for this function:

```cpp
PluginManager::Serializer & Framework::Managers::SerializerManager::Get (const std::string & name) [virtual]
```

Retrieves the specified Serializer.

**Parameters:**
- `name` the name of the Serializer

**Returns:**
a reference to the Serializer
Referenced by Deserialize(), and Serialize().

Here is the caller graph for this function:

```cpp
void Framework::Managers::SerializerManager::Serialize (const std::string & format, const PluginManager::Object & object, const PluginManager::Mode_t & mode, const std::string & filename) [virtual]
```

Converts the Object to a string in the second format in the specified mode.

**Parameters:**
- `format` The format with which to encode the Object
- `object` The Object to encode
- `mode` FULL encodes every non-null field of the Object. CHANGES encodes only the fields marked as changed
- `filename` The file in which to store the encoded Object
References Serialize().

Here is the call graph for this function:

std::string Framework::Managers::SerializerManager::Serialize (const std::string & format, const PluginManager::Object & object, const PluginManager::Mode_t & mode) [virtual]

Converts the Object to a string in the second format in the specified mode.

Parameters:
- format The format with which to encode the Object
- object The Object to encode
- mode FULL encodes every non-null field of the Object. CHANGES encodes only the fields marked as changed

Returns:
the encoded text

References Get(), and PluginManager::Serializer::SetMode().
Referenced by Serialize(), and SerializerManager().

Here is the call graph for this function:

The documentation for this class was generated from the following files:
- SerializerManager.hpp
- SerializerManager.cpp

Yoohoo::Session Class Reference

Session translates higher level C++ calls into XML for transmission to remote hosts and translates inbound XML from remote hosts into C++ processable signals.
#include <Session.hpp>
Inherits Networking::Session.
Inherited by Insight::HostSynchronizer::Session.
Collaboration diagram for Yoohoo::Session:

Public Member Functions
- Session (boost::asio::io_service &ioService, const std::string &host, const int port)
  Constructor. Used to initiate outbound connections.
- Session (boost::asio::io_service &ioService)
  Constructor. Used to receive inbound connections.
virtual void Yoohoo(const std::string &hostId, const std::string &host, const int &port)
Sends information necessary to make a return connection to this host.

virtual void Discover()  
Requests a list of hosts from the remote host.

virtual void Host(const std::string &hostId, const std::string &host, const int &port)
Alerts the remote host to the existence of a third host.

virtual void Hosts(const std::map<std::string, std::pair<std::string, int>> &hosts)
Alerts the remote host to the existence of multiple hosts.

Public Attributes
boost::signals2::signal<void(boost::shared_ptr<Networking::Session> session, const std::string &hostId, const std::string &host, const int &port)> signal_yoohoo
boost::signals2::signal<void(boost::shared_ptr<Networking::Session> session)> signal_discover
boost::signals2::signal<void(boost::shared_ptr<Networking::Session> session, const std::string &hostId, const std::string &host, const int &port)> signal_host
boost::signals2::signal<void(boost::shared_ptr<Networking::Session> session, const std::map<std::string, std::pair<std::string, int>> &hosts)> signal_hosts

Protected Member Functions
virtual void Parse(const std::string &body)
Parses an incoming message once the entire message has been received.

Detailed Description
Session translates higher level C++ calls into XML for transmission to remote hosts and translates inbound XML from remote hosts into C++ processable signals.

Constructor & Destructor Documentation
Yoohoo::Session::Session(boost::asio::io_service & ioService, const std::string & host, const int port)

Constructor. Used to initiate outbound connections.

Todo:
add SSL option

Parameters:
ioService
host
port

Returns:
Reimplemented from Networking::Session (p.285).

Yoohoo::Session::Session(boost::asio::io_service & ioService)
Constructor. Used to receive inbound connections.

Todo:
- add SSL option

Parameters:
- ioService

Reimplemented from Networking::Session (p. 286).

---

Member Function Documentation

void Yoohoo::Session::Discover () [virtual]

Requests a list of hosts from the remote host.

Precondition:
The YOOHOO handshake has been completed.
A host sends the DISCOVER command to determine what other hosts exist. When a host receives DISCOVER, sends the HOSTS command to the remote host.

References Networking::Session::Send().

Here is the call graph for this function:

```
void Yoohoo::Session::Discover() -> Networking::Session::Send() -> Networking::Session::handle_write()
```

void Yoohoo::Session::Host (const std::string & hostId, const std::string & host, const int & port) [virtual]

Alerts the remote host to the existence of a third host.

A host MAY issue the HOST command its connected hosts up receipt of YOOHOO.

Parameters:
- `hostId` A unique identifier specifying the third host
- `host` A routable host name or ip address of the third host
- `port` The port on which the third host is listening for incoming connections

References Networking::Session::Send().

Here is the call graph for this function:

```
void Yoohoo::Session::Host() -> Networking::Session::Send() -> Networking::Session::handle_write()
```

void Yoohoo::Session::Hosts (const std::map< std::string, std::pair< std::string, int > > & hosts) [virtual]

Alerts the remote host to the existence of multiple hosts.
When HOSTS is received, the host MUST attempt a YOOHOO to each received host
Parameters:

hosts map keyed on hostId containing host name, port number pairs

References Networking::Session::Send().

Here is the call graph for this function:

![Call Graph]

void Yoohoo::Session::Parse (const std::string & body) [protected, virtual]

Parses an incoming message once the entire message has been received.

Parameters:

body

Implements Networking::Session (p.287).

Reimplemented in Insight::HostSynchronizer::Session (p.284).

void Yoohoo::Session::Yoohoo (const std::string & hostId, const std::string & host, const int & port) [virtual]

Sends information necessary to make a return connection to this host.

When a YOOHOO is received, the local host MUST check its hosts table to determine if it is already connected to the remote host. If it is, no further action need be taken. If it is not, the remote host MUST be added to the hosts table and a YOOHOO MUST be sent back to the remote host. This will effectively result in a two and half way handshake where in the initiating host sends YOOHOO twice and the accepting sends YOOHOO once.

Parameters:

hostId A unique identifier specifying this host
host A routable host name or ip address
port The port on which this host is listening for incoming connections

References Networking::Session::Send().

Here is the call graph for this function:

![Call Graph]

Member Data Documentation

boost::signals2::signal<void(boost::shared_ptr<Networking::Session> session)>
Yoohoo::Session::signal_discover

Emitted when a DISCOVER is received

boost::signals2::signal<void(boost::shared_ptr<Networking::Session> session, const std::string& hostId, const std::string& host, const int& port)>
Yoohoo::Session::signal_host

Emitted when a HOST is received
Parameters:
  - hostId
  - host
  - port

```cpp
boost::signals2::signal<void(boost::shared_ptr<Networking::Session> session, const std::map<std::string, std::pair<std::string, int>> & hosts)>
  Yoohoo::Session::signal_hosts
```
Emitted when a HOSTS is received

Parameters:
  - hosts

```cpp
boost::signals2::signal<void(boost::shared_ptr<Networking::Session> session, const std::string & hostId, const std::string & host, const int & port)>
  Yoohoo::Session::signal_yoohoo
```
Emitted when a YOOHOO is received

Parameters:
  - hostId
  - host
  - port

The documentation for this class was generated from the following files:
- `insight_plugin_host_synchronizer/src/Yoohoo/Session.hpp`
- `insight_plugin_host_synchronizer/src/Yoohoo/Session.cpp`

---

**Insight::Utility::Session Class Reference**

Contains the low level functions of the **POP3** protocol specified in RFC 1939.

```cpp
#include <Session.hpp>
```

Collaboration diagram for Insight::Utility::Session:
Classes

- class InvalidResponse
  Thrown by GetResponse() if the server response cannot be interpreted. class InvalidStateException
  thrown by any POP3 command function that is called when the server is in a state that does not allow said function. class NegativeResponse

Thrown by GetResponse() if the server response begins with "-ERR". Public Member Functions

- Session (PluginManager::PlatformServices &services, const std::string &server, const Security_t &security, const int port=110)
  Constructor Initializes IoService with Socket. Connects Socket to server on port.
- virtual ~Session ()
  Destructor. Closes the socket.
- virtual void Quit ()
  Valid during the TRANSACTION state and the AUTHORIZATION state after USER has failed. The command tells the server close the TCP connection. If the server is in the TRANSACTION state, the server enters the UPDATE state.
- virtual const std::pair< int, int > Stat () const
  Retrieves a drop listing for the maildrop.
- virtual const std::vector< int > List () const
- virtual const int List (const int msg) const
  Retrieves a scan listing for the specified message.
- virtual const std::string Retr (const int msg) const
  After the initial +OK, the POP3 server sends the message corresponding to the given message-number.
- virtual void Dele (const int msg) const
  The POP3 server marks the message as deleted. Any future reference to the message-number associated with the message in a POP3 command generates an error. The POP3 server does not actually delete the message until the POP3 session enters the UPDATE state.
- virtual void Noop () const
  The POP3 server does nothing, it merely replies with a positive response.
- virtual void Rset () const
  If any messages have been marked as deleted by the POP3 server, they are unmarked. The POP3 server then replies with a positive response.
- virtual const std::string Top (const int msg, const size_t lines=0) const
  The POP3 server sends the headers of the message, the blank line separating the headers from the body, and then the number of lines of the indicated message’s body, being careful to byte-stuff the termination character (as with all multi-line responses).
- virtual const std::string Uidl (const int msg) const
  the POP3 server issues a positive response with a line containing information for that message. This line is called a “unique-id listing” for that message.
- virtual const std::vector< std::string > Uidl () const
  the POP3 server issues a positive response with a line containing information for that message. This line is called a "unique-id listing" for that message.
- virtual void User (const std::string &name)
  Specifies the Maildrop for which to authenticate this session.
- virtual void Pass (const std::string &password)
Supplies a password to accompany the maildrop specified in the USER command.

- virtual void Apop (const std::string &name, const std::string &digest)
  Valid during the AUTHORIZATION state after the POP3 greeting or after an unsuccessful USER or PASS command.

- virtual const std::map< std::string, std::string > Capa ()
  Valid during the AUTHORIZATION and TRANSACTION states.

Protected Member Functions

- virtual void SendRequest (const std::string &request) const
- virtual std::string GetResponse (const bool multiline) const

Protected Attributes

- StateType State
- bool UserSuccessful
- bool UserFailed

Detailed Description

Contains the low level functions of the POP3 protocol specified in RFC 1939.

Constructor & Destructor Documentation

Insight::Utility::Session::Session (PluginManager::PlatformServices & services, const std::string & server, const Security_t & security, const int port = 110)

Constructor Initializes IoService with Socket. Connects Socket to server on port.

Parameters:

- services A pointer to the PlatformServices struct
- server the DNS name or IP address of the POP3 server
- port the port on which to connect to the POP3 server
- security the security method to use

Exceptions:

@throws

server not ready
server not available

References GetResponse(), NetworkInterface::MakeConnection(), State, UserFailed, and UserSuccessful.

Here is the call graph for this function:
Insight::Utility::Session::~Session () [virtual]

Destructor. Closes the socket.

---

**Member Function Documentation**

virtual void Insight::Utility::Session::Apop (const std::string & name,   const std::string & digest) [inline, virtual]

Valid during the AUTHORIZATION state after the POP3 greeting or after an unsuccessful USER or PASS command.

**Parameters:**
- name a string identifying a mailbox
- digest an MD5 digest string

**Exceptions:**
- **InvalidState** if not in the AUTHORIZATION state
- **std::invalid_argument** if the server returns a negative response

const std::map< std::string, std::string > Insight::Utility::Session::Capa () [virtual]

Valid during the AUTHORIZATION and TRANSACTION states.

**Returns:**
- a list of the server's capabilities

**Exceptions:**
- **NegativeResponse** if the server does not support the CAPA command

References GetResponse(), SendRequest(), and State.

Here is the call graph for this function:

```plaintext
Insight::Utility::Session::Capa
  ^                        ^
  |                        |
  |                        |
  v                        v
Insight::Utility::Session::GetResponse
Insight::Utility::Session::SendRequest
```

void Insight::Utility::Session::Dele (const int msg) const [virtual]

The POP3 server marks the message as deleted. Any future reference to the message-number associated with the message in a POP3 command generates an error. The POP3 server does not actually delete the message until the POP3 session enters the UPDATE state.

**Parameters:**
- msg a message-number (required) which may NOT refer to a message marked as deleted
Exceptions:

- std::invalid_argument if msg is not found on the server
- InvalidState if not in the TRANSACTION state

References GetResponse(), SendRequest(), and State.

Here is the call graph for this function:

```
string Insight::Utility::Session::GetResponse (const bool multiline) const [protected, virtual]
```

Here is the caller graph for this function:
const int Insight::Utility::Session::List (const int msg) const  [virtual]

Retrieves a scan listing for the specified message.

Parameters:

msg a message-number (optional), which, if present, may NOT refer to a message marked as deleted

Returns:

the size of the msg in octets
Exceptions:

- \texttt{std::invalid	extunderscore argument} if msg does not exist on the server
- \texttt{InvalidState} if not in the TRANSACTION state

add code to confirm that the pattern was found?

References \texttt{GetResponse()}, \texttt{SendRequest()}, and \texttt{State}.

Here is the call graph for this function:

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{call_graph.png}
\caption{Call graph for the function \texttt{Insight::Utility::Session::List}.

\textbf{const vector< int > \texttt{Insight::Utility::Session::List} () const} \texttt{[virtual]}

Retrieves a scan listing for each message on the server.

\textbf{Returns:}

- a vector containing the size in octets of each message on the server

Exceptions:
- \texttt{InvalidState} if not in the TRANSACTION state

References \texttt{GetResponse()}, \texttt{SendRequest()}, and \texttt{State}.

Here is the call graph for this function:

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{call_graph.png}
\caption{Call graph for the function \texttt{Insight::Utility::Session::List}.

\textbf{void \texttt{Insight::Utility::Session::Noop} () const} \texttt{[virtual]}

The \texttt{POP3} server does nothing, it merely replies with a positive response.

Exceptions:
- \texttt{InvalidState} if not in the TRANSACTION state

References \texttt{GetResponse()}, \texttt{SendRequest()}, and \texttt{State}.

Here is the call graph for this function:

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{call_graph.png}
\caption{Call graph for the function \texttt{Insight::Utility::Session::Noop}.

\textbf{void \texttt{Insight::Utility::Session::Pass} (const std::string & \texttt{password})} \texttt{[virtual]}

\texttt{}}
Supplies a password to accompany the maildrop specified in the USER command.

**Parameters:**
- *password* a server/mailbox-specific password (required)

**Exceptions:**
- *InvalidState* if not in the AUTHORIZATION state and UserSuccessful is not true
- *std::invalid_argument* if the server returns a negative response

References GetResponse(), SendRequest(), State, and UserSuccessful.

Here is the call graph for this function:

```plaintext
Insight::Utility::Session::Pass -> Insight::Utility::Session::GetResponse
Insight::Utility::Session::SendRequest
```

**void Insight::Utility::Session::Quit () [virtual]**

Valid during the TRANSACTION state and the AUTHORIZATION state after USER has failed. The command tells the server close the TCP connection. If the server is in the TRANSACTION state, the server enters the UPDATE state.

**Exceptions:**
- *NegativeResponse* if messages could not be deleted in the TRANSACTION state.
- *InvalidState* if called in the UPDATE state or AUTHORIZATION state and UserFailed and UserSuccessful are both true. if the QUIT command is issued twice in the same session.

References GetResponse(), SendRequest(), State, UserFailed, and UserSuccessful.

Here is the call graph for this function:

```plaintext
Insight::Utility::Session::Quit -> Insight::Utility::Session::GetResponse
Insight::Utility::Session::SendRequest
```

**const string Insight::Utility::Session::Retr (const int msg) const [virtual]**

After the initial +OK, the POP3 server sends the message corresponding to the given message-number.

**Parameters:**
- *msg* a message-number (required) which may NOT refer to a message marked as deleted

**Returns:**
- a raw message in the form of a string

**Exceptions:**
- *std::invalid_argument* if msg is not found on the server
- *InvalidState* if not in the TRANSACTION state
References GetResponse(), SendRequest(), and State.
Here is the call graph for this function:

```
void Insight::Utility::Session::Rset () const  [virtual]
```

If any messages have been marked as deleted by the POP3 server, they are unmarked. The POP3 server then replies with a positive response.

**Exceptions:**
- `InvalidState` if not in the TRANSACTION state

References GetResponse(), SendRequest(), and State.
Here is the call graph for this function:

```
void Insight::Utility::Session::SendRequest (const std::string & request) const  [protected, virtual]
```

Sends a request to the server.

**Parameters:**
- `request` the string to be sent to the server. Does not include the required CRLF

References PluginManager::PlatformServices::LogEvent.

Referenced by Capa(), Dele(), List(), Noop(), Pass(), Quit(), Retr(), Rset(), Stat(), Top(), Uidl(), and User().

Here is the caller graph for this function:
const pair<int, int> Insight::Utility::Session::Stat() const [virtual]

Retrieves a drop listing for the maildrop.

**Returns:**

FIRST: the number of messages in the maildrop (does not include deleted messages)
SECOND: the size of the maildrop in octets (does not include deleted messages)

**Exceptions:**

*InvalidState* if not in the TRANSACTION state

add code to confirm that the pattern was found?
References GetResponse(), SendRequest(), and State.
Here is the call graph for this function:

```plaintext
const string Insight::Utility::Session::Top (const int msg, const size_t lines = 0) const [virtual]
```

The POP3 server sends the headers of the message, the blank line separating the headers from the body, and then the number of lines of the indicated message’s body, being careful to byte-stuff the termination character (as with all multi-line responses).

**Parameters:**

- `msg` a message-number (required) which may NOT refer to to a message marked as deleted, and a non-negative number of lines (required)

**Returns:**

- a raw message in the form of a string

**Exceptions:**

- `std::invalid_argument` if `msg` is in `DeletedMessages` or is not on server
- `InvalidState` if not in the TRANSACTION state

add some intelligence to the Pop3 class to disable header retrieval if it can be confirmed that `msg` does indeed exist

References GetResponse(), SendRequest(), and State.
Here is the call graph for this function:

```plaintext
const vector< string > Insight::Utility::Session::Uidl () const [virtual]
```

the POP3 server issues a positive response with a line containing information for that message. This line is called a "unique-id listing" for that message.

**Returns:**

- a vector containing the unique-id of each message on the server

**Exceptions:**

- `InvalidState` if not in the TRANSACTION state

References GetResponse(), SendRequest(), and State.
const string Insight::Utility::Session::Uidl (const int msg) const [virtual]

the POP3 server issues a positive response with a line containing information for that message. This line is called a "unique-id listing" for that message.

Parameters:
msg a message-number (optional), which, if present, may NOT refer to a message marked as deleted

Returns:
the unique-id of the specified message

Exceptions:
std::invalid_argument if msg is in DeletedMessages or is not on server
InvalidState if not in the TRANSACTION state
References GetResponse(), SendRequest(), and State.

void Insight::Utility::Session::User (const std::string & name) [virtual]

Specifies the Maildrop for which to authenticate this session.

Parameters:
name a string identifying a mailbox (required), which is of significance ONLY to the server. Sets either UserSuccessful or UserFailed to true

Exceptions:
InvalidState if not in the AUTHORIZATION state.
std::invalid_argument if the server returns a negative response
References GetResponse(), SendRequest(), State, UserFailed, and UserSuccessful.
Member Data Documentation

**StateType** Insight::Utility::Session::State  [protected]
Stores the current state of the server
Referenced by Capa(), Dele(), List(), Noop(), Pass(), Quit(), Retr(), Rset(), Session(), Stat(), Top(), Uidl(), and User().

**bool** Insight::Utility::Session::UserFailed  [protected]
set to TRUE if USER fails
Referenced by Quit(), Session(), and User().

**bool** Insight::Utility::Session::UserSuccessful  [protected]
set to TRUE if USER executes successfully
Referenced by Pass(), Quit(), Session(), and User().

The documentation for this class was generated from the following files:
- insight_plugin_pop3/Session.hpp
- insight_plugin_pop3/Session.cpp

---

**Insight::HostSynchronizer::Session Class Reference**

Extends Yoohoo::Session to add synchronization commands.
#include <Session.hpp>
Inherits Yoohoo::Session.
Collaboration diagram for Insight::HostSynchronizer::Session:
Public Member Functions

- **Session** (PluginManager::PlatformServices &services, boost::asio::io_service &ioService, const std::string &host, const int port)
  Constructor. Used to initiate outbound connections.

- **Session** (PluginManager::PlatformServices &services, boost::asio::io_service &ioService)
  Constructor. Used to handle incoming connections.

- virtual void **SubmitKeyChange** (const std::string &oldKey, const std::string &newKey)

Protected Member Functions

- virtual void **Send** (const std::string &message)
  Asynchronously sends message on Socket.

- virtual void **Parse** (const std::string &message)
  Parses an incoming message once the entire message has been received.

Detailed Description

Extends **Yoohoo::Session** to add synchronization commands.

Constructor & Destructor Documentation

**Insight::HostSynchronizer::Session::Session** (PluginManager::PlatformServices & services, boost::asio::io_service & ioService, const std::string & host, const int port)

Constructor. Used to initiate outbound connections.

**Parameters:**
- services
- ioService
- host The hostname or ip address to which to connect
- port The port on which to connect to the remote host

**Insight::HostSynchronizer::Session::Session** (PluginManager::PlatformServices & services, boost::asio::io_service & ioService)

Constructor. Used to handle incoming connections.

**Parameters:**
- services
- ioService
Member Function Documentation

```cpp
void Insight::HostSynchronizer::Session::Parse (const std::string & body) [protected, virtual]
```

Parses an incoming message once the entire message has been received.

**Parameters:**

- `body`

Reimplemented from `Yoohoo::Session (p.267)`.

References `PluginManager::PlatformServices::LogEvent`.

```cpp
void Insight::HostSynchronizer::Session::Send (const std::string & message) [protected, virtual]
```

Asynchronously sends message on Socket.

**Parameters:**

- `message`

Reimplemented from `Networking::Session (p.288)`.

References `PluginManager::PlatformServices::LogEvent`.

Referenced by `SubmitKeyChange()`.

Here is the caller graph for this function:

```
Insight::HostSynchronizer::Session::Send -> Insight::HostSynchronizer::Session::SubmitKeyChange
```

```cpp
void Insight::HostSynchronizer::Session::SubmitKeyChange (const std::string & oldKey, const std::string & newKey) [virtual]
```

Note: this may not work properly in offline mode

**Parameters:**

- `oldKey`
- `newKey`

References `Send()`.

Here is the call graph for this function:

```
Insight::HostSynchronizer::Session::SubmitKeyChange -> Insight::HostSynchronizer::Session::Send
```

---

The documentation for this class was generated from the following files:

- `insight_plugin_host_synchronizer/src/HostSynchronizer/Session.hpp`
- `insight_plugin_host_synchronizer/src/HostSynchronizer/Session.cpp`
Networking::Session Class Reference

The Session class provides a base from which to derive asynchronous protocols.

```cpp
#include <Session.hpp>
```

Inherits `boost::enable_shared_from_this< Session >`.

Inherited by Yoohoo::Session.

### Public Member Functions

- **Session** (boost::asio::io_service &ioService, const std::string &host, const int port)
  Constructor. Used to initiate outbound connections.

- **Session** (boost::asio::io_service &ioService)
  Constructor. Used to receive inbound connections.

- **virtual ~Session** ()
  Destructor. Closes the connection, if necessary.

- **virtual void Start** ()
  Starts an asynchronous read operation.

- **virtual boost::asio::ip::tcp::socket & GetSocket** ()
  Retrieves the underlying boost::asio::ip::tcp::socket.

### Protected Member Functions

- **virtual void Send** (const std::string &message)
  Asynchronously sends message on Socket.

- **virtual void handle_read** (const boost::system::error_code &error)
  Start() completion handler.

- **virtual void handle_write** (const boost::system::error_code &error)
  Send() completion handler.

- **virtual void Parse** (const std::string &body)=0
  Parses an incoming message once the entire message has been received.

### Detailed Description

The Session class provides a base from which to derive asynchronous protocols.

### Constructor & Destructor Documentation

**Networking::Session::Session** (boost::asio::io_service & ioService, const std::string & host, const int port)

Constructor. Used to initiate outbound connections.

**Todo:**
- add SSL option
Parameters:
  `ioService`
  `host`
  `port`

Returns:

Reimplemented in `Yoohoo::Session (p.265)`. References Start().

Here is the call graph for this function:

```
Networking::Session::Session (boost::asio::io_service & `ioService`)  
Networking::Session::Start  
Networking::Session::handle_read  
Networking::Session::Parse  
```

Networking::Session::Session (boost::asio::io_service & `ioService`)

Constructor. Used to receive inbound connections.

Todo:
  add SSL option

Parameters:
  `ioService`

Reimplemented in `Yoohoo::Session (p.265)`.

Networking::Session::~Session () [virtual]

Destructor. Closes the connection, if necessary.

Member Function Documentation

ip::tcp::socket & Networking::Session::GetSocket () [virtual]

Retrieves the underlying boost::asio::ip::tcp::socket.

Returns:
  A reference to Socket

void Networking::Session::handle_read (const boost::system::error_code & `error`) [protected, virtual]

Start() completion handler.
Called when an asynchronous read returns. Since the full message may not be downloaded yet, checks the message to see the end sequence has been received. If it has, passes the message to Parse. In either case, calls start at completion. It should be noted that the current implementation may alter white space.

Parameters:
  `error`
References Parse(), and Start().
Referenced by Start().
Here is the call graph for this function:

```
Networking::Session::Parse
```

```
Networking::Session::handle_read
Networking::Session::Start
```

void Networking::Session::handle_write (const boost::system::error_code & error) [protected, virtual]

Send() completion handler.

**Parameters:**
* error
Referenced by Send().
Here is the caller graph for this function:

```
Networking::Session::handle_write
```

```
Networking::Session::Send
```

```
Networking::Session::Discover
```

```
Networking::Session::Hosts
```

```
Networking::Session::Yoohoo
```

virtual void Networking::Session::Parse (const std::string & body) [protected, pure virtual]

Parses an incoming message once the entire message has been received.

**Parameters:**
* body
Implemented in Insight::HostSynchronizer::Session (p.284), and Yoohoo::Session (p.267).
Referenced by handle_read().
Here is the caller graph for this function:

```
Networking::Session::Parse
Networking::Session::handle_read
Networking::Session::Start
```

```
Networking::Session::Session
```

```
Insight::HostSynchronizer::Session
```

```
Yoohoo::Session
```
void Networking::Session::Send (const std::string & message) [protected, virtual]

Asynchronously sends message on Socket.

Parameters:

message
Reimplemented in Insight::HostSynchronizer::Session (p.284).
References handle_write().
Referenced by Yoohoo::Session::Discover(), Yoohoo::Session::Host(), Yoohoo::Session::Hosts(), and Yoohoo::Session::Yoohoo().
Here is the call graph for this function:

Here is the caller graph for this function:

void Networking::Session::Start () [virtual]

Starts an asynchronous read operation.

Precondition:

Socket is connected to a remote host
References handle_read().
Referenced by handle_read(), and Session().
Here is the call graph for this function:

Here is the caller graph for this function:
The documentation for this class was generated from the following files:

- insight_plugin_host_synchronizer/src/Networking/Session.hpp
- insight_plugin_host_synchronizer/src/Networking/Session.cpp

---

**Insight::MySQL::SQLTransaction Class Reference**

provides a base class from which to specialize SQL transactions.

```cpp
#include <SQLTransaction.hpp>
```

Inherits **DAL::Transaction**.
Inherited by **Insight::MySQL::Transactions::Define**, **Insight::MySQL::Transactions::Exec**, **Insight::MySQL::Transactions::Insert**, **Insight::MySQL::Transactions::Query**, **Insight::MySQL::Transactions::Select**, and **Insight::MySQL::Transactions::Update**.

Collaboration diagram for Insight::MySQL::SQLTransaction:
Public Member Functions

- **SQLTransaction** (const std::string &database, const std::string &server, const std::string &username, const std::string &password, PluginManager::PlatformServices &services)
  
  Connects to the database server and enables multiline SQL statements.

Protected Member Functions

- virtual void **DoExec** (const std::string &sql)
  
  Executes SQL that does not return a result.

- virtual PluginManager::Backend::QueryResult **DoQuery** (const std::string &sql)
  
  Executes SQL that returns a result.

- std::pair< std::string, std::string > **Expand** (const DAL::FieldSubMap &fields, const std::string &group="")
  
  Converts a table's field from variables to the pair of strings for the "SELECT " and "FROM " lines.

- std::string **Rubberize** (const DAL::FieldBase &field)
  
  Adds escape characters and, if necessary, enclosing parentheses.

- std::string **Rubberize** (const std::string &field)
  
  Adds escape characters and, if necessary, enclosing parentheses.

Detailed Description

Provides a base class from which to specialize SQL transactions.

This class defines functions for running SQL queries and executing SQL statements. The functions are utilized by derived classes.

Constructor & Destructor Documentation

**Insight::MySQL::SQLTransaction::SQLTransaction** (const std::string &database, const std::string &server, const std::string &username, const std::string &password, PluginManager::PlatformServices &services)

Connects to the database server and enables multiline SQL statements.

Parameters:

- **database** The name of the database with which to work
- **server** the host name or ip address at which the database resides
- **username** the database username
- **password** the database user password
- **services** a reference to the PlatformServices struct

Member Function Documentation

**void Insight::MySQL::SQLTransaction::DoExec** (const std::string &sql) [protected, virtual]
Executes SQL that does not return a result.

**Parameters:**

- `sql`

**PluginManager::Backend::QueryResult**

```cpp
In::MySQL::SQLTransaction::DoQuery (const std::string & sql) [protected, virtual]
```

Executes SQL that returns a result.

**Parameters:**

- `sql`

**Returns:**

std::pair<std::string, std::string> Insight::MySQL::SQLTransaction::Expand (const DAL::FieldSubMap & fields, const std::string & group = "") [protected]

converts a table's field from variables to the pair of strings for the "SELECT " and "FROM " lines

**Parameters:**

- `fields`
- `group`

**Returns:**

std::string Insight::MySQL::SQLTransaction::Rubberize (const std::string & field) [protected]

adds escape characters and, if necessary, enclosing paranthises

**Parameters:**

- `field`

**Returns:**

std::string Insight::MySQL::SQLTransaction::Rubberize (const DAL::FieldBase & field) [protected]

adds escape characters and, if necessary, enclosing paranthises

**Parameters:**

- `field`
The documentation for this class was generated from the following files:

- SQLTransaction.hpp
- SQLTransaction.cpp

**DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE > Class Template Reference**

Wrapper template from which other Collections are derived/specialized. Amongst other things, prodides wrappers for STL iterators so that they can work with rest of DAL.

```cpp
#include <STLCollection.hpp>
```

Inherits DAL::Collection, and DAL::Transaction.

Inherited by DAL::Collections::Sequence< VAL_TYPE >, DAL::Collections::Sequence< Reference >, and DAL::Collections::Sequence< std::string >.

Collaboration diagram for DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >:
null
Public Member Functions

- virtual void **for_each** (boost::function< void(FieldMap &fields)> functor)
- virtual void **for_each** (boost::function< void(ConstFieldMap &fields)> functor) const
- virtual bool **IsAltered** () const
  Indicates if any of Collections fields have been changed since the last database interaction.
- virtual void **SetAltered** (const bool &altered)
  Can be used to explicitly override the altered state (mostly used by Transactions after they have done their work.
- STL_TYPE & **_impl** ()
  provides access to the underlying STL container
- const STL_TYPE & **_impl** () const
  provides access to the underlying STL container
- iterator **begin** ()
- const_iterator **begin** () const
- iterator **end** ()
- const_iterator **end** () const
- reverse_iterator **rbegin** ()
- const_reverse_iterator **rbegin** () const
- reverse_iterator **rend** ()
- const_reverse_iterator **rend** () const
- virtual bool **empty** () const
  STL Wraper Function.
- virtual size_t **size** () const
  STL Wraper Function.
- virtual size_t **max_size** () const
  STL Wraper Function.
- template<class InputIterator > void **insert** (iterator position, InputIterator first, InputIterator last)
  Inserts the range [first, last] at position.
- virtual void **erase** (iterator position)
  erases the item at the specified position
- virtual void **erase** (iterator first, iterator last)
  erases the items in the range [first, last]
- virtual void **swap** (STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE > &other)
  Swaps the (*this).Container with other.
- virtual void **clear** ()
  Clears the underlying container.
- virtual STLCollection & **operator=** (const STL_TYPE &other)
  Copies other this this->Container.
Detailed Description

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> class DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >

Wrapper template from which other Collections are derived/specialized. Amongst other things, provides wrappers for STL iterators so that they can work with rest of DAL.

Member Function Documentation

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> const STL_TYPE& DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::_impl () const [inline]

provides access to the underlying STL container

Returns:
the underlying STL container

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> STL_TYPE& DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::_impl () [inline]

provides access to the underlying STL container

Returns:
the underlying STL container

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> const_iterator DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::begin () const [inline]

STL function

Returns:
An iterator to the first element

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> iterator DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::begin () [inline]

STL function

Returns:
An iterator to the first element
Referenced by DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, const std::pair< const std::string, std::string > >::forEach(), DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, const std::pair< const std::string, std::string > >::IsAltered(), DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, const std::pair< const std::string, std::string > >
>, const std::pair< const std::string, std::string > >::SetAltered(), and
Insight::InternetMessageFormat::Message::ToNativeFormat().
Here is the caller graph for this function:

\[
\text{DAL::STLCollection::clear () [inline, virtual]}
\]

Clears the underlying container.
Implements DAL::Collection (p.92).

\[
\text{template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename}
\text{CONST_DEREFERENCE_TYPE> virtual void DAL::STLCollection<}
\text{STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE} >::clear () [inline, virtual]
\]

STL Wraper Function.

\textbf{Returns:}

true if there are no items in the Collection, false otherwise
Implements DAL::Collection (p.92).
Referenced by Insight::InternetMessageFormat::Message::ToNativeFormat().
Here is the caller graph for this function:

\[
\text{DAL::STLCollection::empty} \rightarrow \text{Insight::InternetMessageFormat::Message::ToNativeFormat}
\]

\[
\text{template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename}
\text{CONST_DEREFERENCE_TYPE> virtual bool DAL::STLCollection<}
\text{STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE} >::empty () const [inline, virtual]
\]

STL Wraper Function.

\textbf{Returns:}

An iterator to the element after the last element

\[
\text{template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename}
\text{CONST_DEREFERENCE_TYPE> const_iterator DAL::STLCollection<}
\text{STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE} >::end () const [inline]
\]

STL function

\textbf{Returns:}

An iterator to the element after the last element

\[
\text{template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename}
\text{CONST_DEREFERENCE_TYPE> Iterator DAL::STLCollection<}
\text{STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE} >::end () [inline]
\]

STL function

\textbf{Returns:}

An iterator to the element after the last element
Referenced by DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, const std::pair< const std::string, std::string > >::for_each(), DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, const std::pair< const std::string, std::string > >::IsAltered(), DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, const std::pair< const std::string, std::string > >::SetAltered(), Insight::InternetMessageFormat::Message::SetField(), and Insight::InternetMessageFormat::Message::ToNativeFormat.

Here is the caller graph for this function:

- DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, const std::pair< const std::string, std::string > >::for_each
- DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, const std::pair< const std::string, std::string > >::IsAltered
- DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, std::string >, const std::pair< const std::string, std::string > >::SetAltered
- Insight::InternetMessageFormat::Message::SetField
- Insight::InternetMessageFormat::Message::ToNativeFormat

---

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> virtual void DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::erase (iterator first, iterator last) [inline, virtual]

erases the items in the range [first, last]

Parameters:

- first
- last

---

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> virtual void DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::erase (iterator position) [inline, virtual]

erases the item at the specified position

Parameters:

- position

---

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> virtual void DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::for_each (boost::function< void(ConstFieldMap &fields)> functor) const [inline, virtual]

Iterates of the items in the Collection and calls functor on the each item. A dereferenced Collection iterator (first defined in STLCollection) returns FieldMap.

Parameters:

- functor A boost::function object that can be called on each item in the Collection

Implements DAL::Collection (p. 92).
template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> virtual void DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::for_each (boost::function<void(FieldMap &fields)> functor) [inline, virtual]

Iterates of the items in the Collection and calls functor on the each item. A dereferenced Collection iterator (first defined in STLCollection) returns FieldMap.

**Parameters:**

functor A boost::function object that can be called on each item in the Collection

Implements DAL::Collection (p.93).

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> template<class InputIterator > void DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::insert (iterator position, InputIterator first, InputIterator last) [inline]

Inserts the range [first, last] at position.

**Parameters:**

position
first
last

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> virtual bool DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::IsAltered () const [inline, virtual]

Indicates if any of Collections fields have been changed since the last database interaction.

**Returns:**

true if an alteration has occurred, false otherwise

Implements DAL::Collection (p.94).

Reimplemented in DAL::Collections::Map< KEY_TYPE, VAL_TYPE > (p.162), DAL::Collections::Sequence< VAL_TYPE > (p.251), DAL::Collections::Map< std::string, PhoneNumber > (p.162), DAL::Collections::Map< std::string, std::string > (p.162), DAL::Collections::Sequence< std::string > (p.251), and DAL::Collections::Sequence< Reference > (p.251).

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> virtual size_t DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::max_size () const [inline, virtual]

STL Wraper Function.

**Returns:**

The maximum number of items that the Collection can store

Implements DAL::Collection (p.94).
template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> virtual STLCollection& DAL::STLCollection<STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE>::operator= (const STL_TYPE & other) [inline, virtual]

Copies other this this->Container.

Parameters:

other

Returns:

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> const_reverse_iterator DAL::STLCollection<STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE>::rbegin () const [inline]

STL function

Returns:

A reverse iterator to the last element

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> reverse_iterator DAL::STLCollection<STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE>::rbegin () [inline]

STL function

Returns:

A reverse iterator to the last element

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> const_reverse_iterator DAL::STLCollection<STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE>::rend () const [inline]

STL function

Returns:

a reverse iterator to the element before the first element

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> reverse_iterator DAL::STLCollection<STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE>::rend () [inline]

STL function

Returns:

a reverse iterator to the element before the first element

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename CONST_DEREFERENCE_TYPE> virtual void DAL::STLCollection<STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE>::SetAltered (const bool & altered) [inline, virtual]

Can be used to explicitly override the altered state (mostly used by Transactions after they have done their work.)
Parameters:

   altered The new value for Altered
Implements DAL::Collection (p.95).

Reimplemented in DAL::Collections::Map< KEY_TYPE, VAL_TYPE > (p.163),
DAL::Collections::Sequence< VAL_TYPE > (p.252), DAL::Collections::Map< std::string, PhoneNumber
> (p.163), DAL::Collections::Map< std::string, std::string > (p.163), DAL::Collections::Sequence<
std::string > (p.252), and DAL::Collections::Sequence< Reference > (p.252).

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename
CONST_DEREFERENCE_TYPE> virtual size_t DAL::STLCollection< STL_TYPE,
DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::size () const [inline, virtual]

STL Wraper Function.

Returns:

   the number of items stored in the Collection
Implements DAL::Collection (p.95).

template<typename STL_TYPE, typename DEREFERENCE_TYPE, typename
CONST_DEREFERENCE_TYPE> virtual void DAL::STLCollection< STL_TYPE,
DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >::swap (STLCollection< STL_TYPE,
DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >& other) [inline, virtual]

Swaps the (*this).Container with other.

Parameters:

   other

The documentation for this class was generated from the following file:
- STLCollection.hpp

Framework::Interface::Synchronizer Class Reference

Defines an interface for keep changes synchronized accross clients.
#include <Synchronizer.hpp>

Inherits PluginManager::Listener.
Inherited by Insight::HostSynchronizer::Peer, and
Insight::Synchronization::SocialSynchronizer.

Collaboration diagram for Framework::Interface::Synchronizer:
Public Member Functions

- **Synchronizer** (PluginManager::PlatformServices &services)
  Constructor. Connects signal handlers, Services.
- virtual ~Synchronizer()
  Destructor. Disconnects signal handlers.

Protected Member Functions

- virtual void handle_object_changed(const std::string &primaryKey)
  Encodes the changed fields of the object specified by primaryKey using a Serializer. The encoded string is placed in StagedChanges.
- virtual void handle_object_committed(const std::string &primaryKey)
  Moves a committed encoded Object string from StagedChanges to CommittedChanges to be picked up by DoWork.
- virtual void handle_object_updated(const std::string &primaryKey)
  Calls handle_object_committed.
- virtual void handle_object_deleted(const std::string &primaryKey)
  Relays a deletion to the other members of the Synchronization group.
- virtual void handle_object_not_changed(const std::string &primaryKey)
  Removes a changed encoded Object string from StagedChanges.
- virtual void handle primaryKey_changed(const std::string &oldId, const std::string &newId)
  Updates the PrimaryKey of an Object in StagedChanges or CommittedChanges. Also, relays the change to the Synchronization group.
- void handle_preference_changed(const std::string &module, const std::string &name, const std::string &value)
  Does nothing at this level, but provides option for derived classes to deal with changed preferences.
- virtual void DoWork()=0
  Implemented by plugin. Should push CommittedChanges to Synchronization group.

Protected Attributes

- std::string Serializer

Detailed Description

Defines an interface for keep changes synchronized across clients.

Constructor & Destructor Documentation

Framework::Interface::Synchronizer::Synchronizer (PluginManager::PlatformServices & services)

Constructor. Connects signal handlers, Services.
**Parameters:**

- `services`

  Reimplemented in `Insight::SocialEmailSynchronizer::Synchronizer (p.308)`.

  References `handle_object_changed()`, `handle_object_committed()`, `handle_object_not_changed()`, `handle_object_updated()`, and `handle_primaryKey_changed()`.

  Here is the call graph for this function:

  ![Call Graph](image)

  **Framework::Interface::Synchronizer::~Synchronizer ()** [virtual]

  Destructor. Disconnects signal handlers.

  This function should not return until CommittedChanges is empty.

  Reimplemented in `Insight::SocialEmailSynchronizer::Synchronizer (p.308)`.

**Member Function Documentation**

**virtual void Framework::Interface::Synchronizer::DoWork ()** [protected, pure virtual]

  Implemented by plugin. Should push CommittedChanges to Synchronization group.

  Implemented in `Insight::HostSynchronizer::Peer (p.189)`, and `Insight::SocialEmailSynchronizer::Synchronizer (p.308)`.

**void Framework::Interface::Synchronizer::handle_object_changed (const std::string & primaryKey)** [protected, virtual]

  Encodes the changed fields of the object specified by primaryKey using a Serializer. The encoded string is placed in StagedChanges.

  **Parameters:**

  - `primaryKey` the PrimaryKey of the changed Object

  Reimplemented in `Insight::HostSynchronizer::Peer (p.190)`, and `Insight::SocialEmailSynchronizer::Synchronizer (p.309)`.

  Referenced by Synchronizer().

  Here is the caller graph for this function:
void Framework::Interface::Synchronizer::handle_object_committed (const std::string & primaryKey) [protected, virtual]

Moves a committed encoded Object string from StagedChanges to CommitedChanges to be picked up by DoWork.

**Parameters:**

- **primaryKey** the PrimaryKey of the committed Object

Reimplemented in **Insight::HostSynchronizer::Peer (p. 190).**

References Serializer.

Referenced by Synchronizer().

Here is the caller graph for this function:

void Framework::Interface::Synchronizer::handle_object_deleted (const std::string & primaryKey) [protected, virtual]

relays a deletion to the other members of the Synchronization group

**Parameters:**

- **primaryKey** the PrimaryKey of the deleted Object

Reimplemented in **Insight::HostSynchronizer::Peer (p. 190).**

void Framework::Interface::Synchronizer::handle_object_not_changed (const std::string & primaryKey) [protected, virtual]

Removes a changed encoded Object string from StagedChanges.

**Parameters:**

- **primaryKey** the PrimaryKey of the unchanged Object

Referenced by Synchronizer().

Here is the caller graph for this function:

void Framework::Interface::Synchronizer::handle_object_updated (const std::string & primaryKey) [protected, virtual]

Calls handle_object_committed.
Parameters:

*primaryKey*

Reimplemented in **Insight::HostSynchronizer::Peer** (p.190).

Referenced by Synchronizer().

Here is the caller graph for this function:

```
Framework::Interface::Synchronizer::handle_object_updated

Framework::Interface::Synchronizer::Synchronizer
```

**void Framework::Interface::Synchronizer::handle_preference_changed (const std::string &module, const std::string &name, const std::string &value)** [protected]

Does nothing at this level, but provides option for derived classes to deal with changed preferences.

Parameters:

*module*

*name*

*value*

Reimplemented in **Insight::HostSynchronizer::Peer** (p.191).

**void Framework::Interface::Synchronizer::handle_primaryKey_changed (const std::string &oldId, const std::string &newId)** [protected, virtual]

Updates the PrimaryKey of an Object in StagedChanges or CommittedChanges. Also, relays the change to the Synchronization group.

Parameters:

*oldId*

*newId*

Reimplemented in **Insight::HostSynchronizer::Peer** (p.191).

Referenced by Synchronizer().

Here is the caller graph for this function:

```
Framework::Interface::Synchronizer::handle_primaryKey_changed

Framework::Interface::Synchronizer::Synchronizer
```

**Member Data Documentation**

**std::string Framework::Interface::Synchronizer::Serializer** [protected]

Specifies which Serializer to use to encode the changeset

Referenced by handle_object_committed().
Insight::SocialEmailSynchronizer::Synchronizer Class Reference

Encodes shared Objects and distributes them to other users. Unlike the HostSynchronizer, this plugin uses it's own XML serializer instead of the PluginManager::Serializer interface. The Serializer interface doesn't have the ability to be customized for every field.

#include <Synchronizer.hpp>
Inherits Insight::Synchronization::SocialSynchronizer, and DAL::Transaction.

Collaboration diagram for Insight::SocialEmailSynchronizer::Synchronizer:

```
DAL::Transaction

# GetFields()
# GetFields()
# GetCollections()
# GetCollections()
# SetPrimaryKey()
# SetForeignKey()
# SetNotAltered()
```

```
Insight::SocialEmailSynchronizer::Synchronizer

# Worker
# Done

+ Synchronizer()
+ ~Synchronizer()
# handle_object_changed()
# handle_object_delete_pending()
# handle_message_parsed()
# handle_preference_changed()
# SerializeFunctor()
# DoWork()
```

**Public Member Functions**

- **Synchronizer** (PluginManager::PlatformServices &services)
  
  Constructor. Binds functions and starts the Worker thread.

- virtual ~Synchronizer ()
Destructor. Instructs the Worker thread to terminate and blocks until it completes.

Protected Member Functions
- virtual void handle_object_changed (const std::string &primaryKey)
- virtual void DoWork ()
  Implemented by plugin. Should push CommittedChanges to Synchronization group.

Detailed Description
Encodes shared Objects and distributes them to other users. Unlike the HostSynchronizer, this plugin uses it's own XML serializer instead of the PluginManager::Serializer interface. The Serializer interface doesn't have the ability to be customized for every field.

Constructor & Destructor Documentation

Insight::SocialEmailSynchronizer::Synchronizer::Synchronizer (PluginManager::PlatformServices & services)

Constructor. Binds functions and starts the Worker thread.

Parameters:
- services
  Reimplemented from Framework::Interface::Synchronizer (p.303).

Insight::SocialEmailSynchronizer::Synchronizer::~Synchronizer () [virtual]

Destructor. Instructs the Worker thread to terminate and blocks until it completes.

Returns:
- Reimplemented from Framework::Interface::Synchronizer (p.304).

Member Function Documentation

void Insight::SocialEmailSynchronizer::Synchronizer::DoWork () [protected, virtual]

Implemented by plugin. Should push CommittedChanges to Synchronization group. Implements Framework::Interface::Synchronizer (p.304).
void Insight::SocialEmailSynchronizer::Synchronizer::handle_object_changed (const std::string & primaryKey) [protected, virtual]

Rather than use the XML synchronizer to publish full objects, this function customizes the XML creation to only publish those fields that should be published.

Parameters:

primaryKey
Reimplemented from Framework::Interface::Synchronizer (p.304).

The documentation for this class was generated from the following files:

- insight_plugin_socialemailsynchronizer/src/Synchronizer.hpp
- insight_plugin_socialemailsynchronizer/src/Synchronizer.cpp

---

**Insight::GTKmm::TextDialog Class Reference**

window acts like a Gtk::Dialog, but instead of resizing for the amount of text displayed, it is a fixed size window containing a text buffer for displaying larger amounts of non-editable text.

```
#include <TextDialog.hpp>
```

Inherits Insight::GTKmm::Widget< Gtk::Window >.

**Detailed Description**

window acts like a Gtk::Dialog, but instead of resizing for the amount of text displayed, it is a fixed size window containing a text buffer for displaying larger amounts of non-editable text.

The documentation for this class was generated from the following files:

- TextDialog.hpp
- TextDialog.cpp

---

**DAL::Transaction Class Reference**

As a friend of Class, classes needing programmatic access to a a Class's members must be derived Transaction.

```
#include <Transaction.hpp>
```

Inherited by DAL::STLCollection< STL_TYPE, DEREFERENCE_TYPE, CONST_DEREFERENCE_TYPE >, Insight::GTKmm::Frontend::FirstRunFunctor, Insight::GTKmm::Views::SyncOptions::Fields, Insight::GTKmm::WidgetObject< IG_GTK_BASE >, Insight::MySQL::SQLTransaction, Insight::SocialEmailSynchronizer::Synchronizer, PluginManager::Serializer, DAL::STLCollection< std::map< KEY_TYPE, VAL_TYPE >, std::pair< const KEY_TYPE, VAL_TYPE >, const std::pair< const KEY_TYPE, VAL_TYPE > >, DAL::STLCollection< std::map< std::string, PhoneNumber >, std::pair< const std::string, PhoneNumber >, const std::pair< const std::string, PhoneNumber > >,
PhoneNumber >, const std::pair< const std::string, PhoneNumber > >, 
DAL::STLCollection< std::map< std::string, std::string >, std::pair< const std::string, 
std::string >, const std::pair< const std::string, std::string > >, and 
Insight::GTKmm::WidgetObject< Gtk::Window >.

Protected Member Functions

- virtual FieldMap & GetFields (Class &c)
- virtual ConstFieldMap & GetFields (const Class &c) const
- virtual CollectionMap & GetCollections (Class &c)
- virtual ConstCollectionMap & GetCollections (const Class &c) const
- virtual void SetPrimaryKey (Class &c, const std::string &primaryKey) const
  changes c's primary key
- virtual void SetForeignKey (Collection &c, const std::string &foreignKey)
- virtual void SetNotAltered (Class &c)

Detailed Description

As a friend of Class, classes needing programmatic access to a Class's members must be derived Transaction.

Member Function Documentation

ConstCollectionMap & DAL::Transaction::GetCollections (const Class & c) const [protected, virtual]

Retrieves a reference to c's Collections map

Parameters:
- c

Returns:
- a reference to c's Collections

CollectionMap & DAL::Transaction::GetCollections (Class & c) [protected, virtual]

Retrieves a reference to c's Collections map

Parameters:
- c

Returns:
- a reference to c's Collections
  Referenced by Insight::XML::Serializer::operator().

Here is the caller graph for this function:

DAL::Transaction::GetCollections ─> Insight::XML::Serializer::operator()

ConstFieldMap & DAL::Transaction::GetFields (const Class & c) const [protected, virtual]

Retrieves a reference to c's Fields map
FieldMap & DAL::Transaction::GetFields (Class & c) [protected, virtual]
Retrieves a reference to c's Fields map
Parameters:
c
Returns:
a reference to c's Fields
Referenced by Insight::XML::Serializer::operator().
Here is the caller graph for this function:

void DAL::Transaction::SetForeignKey (Collection & c, const std::string & foreignKey) [protected, virtual]
Assigns foreignKey to c.
Parameters:
c
foreignKey

void DAL::Transaction::SetNotAltered (Class & c) [protected, virtual]
Programmatically sets altered status of every member of c to false
Parameters:
c

void DAL::Transaction::SetPrimaryKey (Class & c, const std::string & primaryKey) const [protected, virtual]
changes c's primary key
Parameters:
c the Class to change
primaryKey the new primary key

The documentation for this class was generated from the following files:
- Transaction.hpp
- Transaction.cpp
Framework::Registry::UnmetDependency Struct Reference

Exception.
#include <Registry.hpp>
Inherits std::exception, and boost::exception.

Detailed Description
Exception.

The documentation for this struct was generated from the following file:
- Registry.hpp

Insight::MySQL::Transactions::Update Class Reference

This functor creates an SQL update script based on the supplied Object.
#include <Update.hpp>
Inherits Insight::MySQL::SQLTransaction.
Collaboration diagram for Insight::MySQL::Transactions::Update:
**Detailed Description**
This functor creates an SQL update script based on the supplied Object.

The documentation for this class was generated from the following files:
- Update.hpp
- Update.cpp

**PluginManager::Version Struct Reference**

Used for storing version numbers.

```cpp
#include <Version.hpp>
```

**Public Member Functions**

- **Version ()**  
  Constructor.
- **Version (const int &major, const int &minor)**  
  Constructor.

**Public Attributes**

- int Major  
  *Major Version Number.*
- int Minor  
  *Minor Version Number.*

**Detailed Description**

Used for storing version numbers.

**Constructor & Destructor Documentation**

**PluginManager::Version::Version () [inline]**

Constructor.

References Major, and Minor.

**PluginManager::Version::Version (const int & major, const int & minor) [inline]**

Constructor.
Member Data Documentation

int PluginManager::Version::Major

Major Version Number.
Referenced by PluginManager::register_plugin(), and Version().

int PluginManager::Version::Minor

Minor Version Number.
Referenced by PluginManager::register_plugin(), and Version().

The documentation for this struct was generated from the following file:
- Version.hpp

Framework::Registry::VersionMismatch Struct Reference

Exception.
#include <Registry.hpp>
Inherits std::exception, and boost::exception.

Detailed Description

Exception.

The documentation for this struct was generated from the following file:
- Registry.hpp