

SFA Modernization Partner
United States Department of Education
Student Financial Assistance



Integrated Technical Architecture
Release 2.0 Technical Specification

Task Order #46
Deliverable # 46.1.3

August 10, 2001



Table of Contents

1 EXECUTIVE SUMMARY	3
1.1 INTRODUCTION	3
1.2 DESCRIPTION OF SECTIONS	4
1.3 APPROACH	4
1.4 SCOPE	4
1.5 ASSUMPTIONS	5
1.6 INTENDED AUDIENCE	5
2 FUNCTIONAL OVERVIEW.....	6
2.1 COMPONENT FACTORY	6
2.2 E-MAIL	6
2.3 EXCEPTION HANDLING	7
2.4 LOGGING	8
2.5 PERSISTENCE	8
2.6 SEARCH.....	9
3 DETAILED DESIGNS.....	10
3.1 COMPONENT FACTORY	10
3.1.1 <i>Introduction</i>	10
3.1.2 <i>System Overview</i>	10
3.1.3 <i>Design Considerations</i>	12
3.1.4 <i>System Architecture</i>	12
3.1.5 <i>Detailed System Design</i>	15
3.1.6 <i>Class Diagrams</i>	26
3.1.7 <i>Interaction Diagram</i>	27
3.1.8 <i>References</i>	28
3.2 E-MAIL	29
3.2.1 <i>Introduction</i>	29
3.2.2 <i>System Overview</i>	29
3.2.3 <i>Design Considerations</i>	29
3.2.4 <i>System Architecture</i>	30
3.2.5 <i>Detailed System Design</i>	31
3.2.6 <i>Class Diagrams</i>	35
3.2.7 <i>Interaction Diagrams</i>	36
3.2.8 <i>References</i>	37
3.3 EXCEPTION HANDLING	38
3.3.1 <i>Introduction</i>	38
3.3.2 <i>System Overview</i>	38
3.3.3 <i>Design Considerations</i>	39
3.3.4 <i>System Architecture</i>	40
3.3.5 <i>Detailed System Design</i>	40



3.3.6	<i>Class Diagrams</i>	49
3.3.7	<i>Interaction Diagrams</i>	50
3.3.8	<i>References</i>	51
3.4	LOGGING.....	52
3.4.1	<i>Introduction</i>	52
3.4.2	<i>System Overview</i>	52
3.4.3	<i>Design Considerations</i>	53
3.4.4	<i>System Architecture</i>	53
3.4.5	<i>Detailed System Design</i>	58
3.4.6	<i>Class Diagrams</i>	101
3.4.7	<i>Interaction Diagrams</i>	104
3.4.8	<i>References</i>	105
3.5	PERSISTENCE	106
3.5.1	<i>Introduction</i>	106
3.5.2	<i>System Overview</i>	106
3.5.3	<i>Design Considerations</i>	108
3.5.4	<i>System Architecture</i>	109
3.5.5	<i>Detailed System Design</i>	111
3.5.6	<i>Class Diagrams</i>	125
3.5.7	<i>Interaction Diagrams</i>	127
3.5.8	<i>References</i>	128
3.6	SEARCH.....	129
3.6.1	<i>Introduction</i>	129
3.6.2	<i>System Overview</i>	129
3.6.3	<i>Design Considerations</i>	129
3.6.4	<i>System Architecture</i>	130
3.6.5	<i>Detailed System Design</i>	134
3.6.6	<i>Class Diagrams</i>	142
3.6.7	<i>Interaction Diagrams</i>	143
3.6.8	<i>References</i>	144
APPENDIX A – ITA RCS AND SFA APPLICATIONS MATRIX		145



1 Executive Summary

1.1 Introduction

The Integrated Technical Architecture (ITA) Release 2.0 (R2.0) builds on Release 1.0 (R1.0) in order to provide the Department of Education's Student Financial Assistance (SFA) program with a more robust core infrastructure for its application and production efforts. In addition to providing SFA's application teams with a core set of products and Subject Matter Expert (SME) support, ITA R2.0 will provide a set of Java 2 Enterprise Edition (J2EE) technical architecture Reusable Common Services (RCS).

The Reusable Common Services (RCS), based completely on open-source technology and Java 2 Enterprise Edition (J2EE), enables solutions for developing multi-tier server-centric applications. Leveraging well established design patterns and extensive experience gained on large-scale enterprise engagements, the RCS provide a robust set of services for application architects to define the fundamental architecture for their application, thereby enabling them to focus on the company's core business.

The main benefits of ITA RCS in a project are as follows:

- Lower cost – Application teams can save hundreds of man-days by not reinventing architectural services.
- Reduced risk – Architecture built from Accenture client experience resulting in robust solutions.
- Speed of Implementation – Allows application developers to focus more on the applications and business logic. It enables application teams to build robust J2EE applications in a shorter timeframe.
- Skillset - Provides some of the reusable architecture services that application project teams do not have or experience to build.
- Quality – Large and growing user base identifies error and problems quickly.

The ITA R2.0 Technical Specification provides information on the following ITA Reusable Common Services:

- Component Factory – Defines a general and extensible factory mechanism. The service enables developers to completely decouple how objects and components are instantiated from their use.
- E-mail Framework – Provides a common way to generate e-mail messages from Java applications.



- Exception Handling Framework – Defines a set of base classes to standardize and simplify exception handling for SFA application teams.
- Logging Framework – Defines a facility to manage and log information messages.
- Persistence Framework – Encapsulates the behavior needed to make objects persistent. Supports saving, deleting, or retrieving many objects at once.
- Search Framework – Simplifies, standardizes, and improves the use of the Autonomy search engine.

1.2 Description of sections

The ITA R2.0 Technical Specification is divided into the following sections:

- Section 1 provides a high level overview of the document.
- Section 2 provides a functional overview of the RCS services.
- Section 3 provides detailed designs for the RCS services.
- Appendix A provides a matrix that details the current and potential usage of RCS services by SFA's applications.

1.3 Approach

The RCS services are built based on an open technology and J2EE architecture. The ITA team leveraged previous Accenture and industry experience to design and develop a robust set of core technical architecture common services that specifically address SFA enterprise application requirements.

The core of the RCS services consists of the exception handling and logging frameworks. The other frameworks use these services to provide standard and consistent exception handling and logging.

1.4 Scope

The ITA Release 2.0 Technical Specification provides information on the components that directly compose the ITA RCS frameworks. While the frameworks make use of many Java features and packages, such as JDBC, RMI, I/O, and JavaMail, the Technical Specification does not cover these topics. For additional information on these topics, please refer to the Sun Java website (<http://www.javasoft.com>) or applicable Java programming guides.

SOAP, UDDI and other services will be explored for ITA Release 3.0. These services as presently evolving standards, and are expected to firm up in the ITA R3.0 timeline.



1.5 Assumptions

The RCS services were built and tested in the ITA R2.0 environment. The following table summarizes the ITA R2.0 products and versions. Products not listed in the table do not interface directly with the ITA R2.0 RCS services consequently upgrades or changes to those products should have no effect on the ITA R2.0 RCS services. While the RCS Services were built using these product versions, the services were built in accordance with J2EE standards and to support product upgrades.

Function	Product	ITA 2.0
HTTP Server	IBM HTTP Server	v. 1.3.12.2
Java Application Server	WebSphere Application Server Advanced Edition	v. 3.5.3
Search Engine	Autonomy Knowledge Server	v. 2.1
Database	Oracle 8i	v. 8.1.6
Java Development Tool	Visual Age for Java Enterprise Edition	v. 3.5.3

1.6 Intended Audience

The ITA R2.0 Technical Specification is intended for ITA and SFA software engineers, standards managers, etc... who need to understand the RCS frameworks in order to troubleshoot or enhance the packages. The document is not intended as a programmer's guide for how to use the frameworks in developing applications. A user's guide detailing how to use the RCS frameworks will be provided in a separate document.



2 Functional Overview

This section provides a functional overview of the following RCS services:

- Component Factory
- E-mail
- Exception Handling
- Logging
- Persistence
- Search

2.1 Component Factory

The component factory defines a general and extensible factory mechanism. The service enables developers to completely decouple how objects and components are instantiated from their use. In addition, the meta-information used to define the object production mechanism and surrounding context is provided via configuration properties, thereby allowing production instances to be changed with minimal effort. The component factory enables the definition of clear migration strategies from one architectural approach to another.

The component factory provides an extensible mechanism to associate producers, targets and context together to produce objects. A common set of producers will be provided that cover a large percentage of scenarios relating to co-located and J2EE object/component production.

This framework will provide the following features:

- Standard methods of producing objects when parameters are passed
- Common parameters to be passed when producing objects
- Standards for coding and designing objects
- Examples for using the component factory framework

2.2 E-mail

The e-mail framework provides SFA with a common way to generate e-mail messages from applications. The e-mail framework uses Sun Microsystems' JavaMail API 1.2, a part of the J2EE framework, which provides a standard interface for Java programs to send e-mails to a Simple Mail Transport Protocol (SMTP) Mail server.

The e-mail framework may be used by SFA application teams to standardize e-mail messaging and replace existing methods of sending e-mail. (The ITA R1.0 applications currently send e-mail



through two methods – one uses JavaMail and the other uses an Oracle database, UNIX shells scripts, and a sendmail operating system utility.)

The ITA e-mail framework provides the following features:

- Dynamically set all elements of an e-mail ("To" address, "From" address, subject, etc.)
- Send attachments to e-mail
- Set multiple e-mail addresses within the "To" address, "From" address, and "Reply To" address
- Dynamically set the SMTP Server
- Verify e-mail parameters meet minimum standards for delivery.
- Send a real time e-mail to a SMTP Server
- Process batch e-mails and send to a SMTP Server at a specified time

2.3 Exception Handling

An exception is a code or language construct that indicates when an unusual or unexpected error condition occurs in an application. Examples of exceptions are hardware, network, I/O, or memory problems. If an exception is “handled” in code, it can be dealt with gracefully and will not necessarily have to cause program termination. Exception handling provides a mechanism for writing robust, resilient code that is capable of dealing with the unexpected.

The exception handling framework will help standardize and simplify exception handling for SFA’s application teams. The standardized exception handling will also help reduce the possibility of uncaught exception scenarios.

The exception handling framework provides the following general services and components:

- Guidelines for identification and responding to exceptions
- Guidelines for throwing and catching exceptions
- Base exception classes
- Default last-resort exception handlers
- Simple interface for integration of logging exceptions
- A generic exception class that must be thrown by all components in the application. It contains a status code that represents the type of exception. This generic exception can be extended for specific errors



- An exception factory class that will be used to create exceptions and will automatically assign a unique id. This unique id will be displayed to the user if necessary in order to uniquely identify the associated log and therefore all the associated information.

2.4 Logging

The logging framework enables users to track and identify the source of errors. The logging framework uses a routine to determine the originating class and method for the logging call. This provides complete and descriptive logs that enable operations personnel to view the logs and quickly determine where the instance or error occurred and possibly what caused it. Logs are not tied to exceptions only; they are customized to record access and other useful information in troubleshooting and analyzing an application.

Users can define handlers to be global or assign them to specific, named loggers. Loggers can be associated to both the global handler set and to specific handlers. The formatting of the message only happens at the handler. Both loggers and handlers can filter messages based on some function, and by the level of the message.

A set of appropriate handlers will be defined for the given application. This could mean Error or higher goes to a specific handler and Info and higher goes to another. In general, these handlers will be global. Application code should log to a named logger appropriate to its subsystem. Finer control of log levels is set within the handler. However, further filtering will require the creation of a custom filter that is attached to the handler.

The logging framework provides the following features:

- Custom logging
- Filtering messages by level
- Integration with the exception handling framework
- Channel functionality to provide the ability to listen to multiple applications on one server

2.5 Persistence

The persistence framework encapsulates the behavior needed to make objects persistent. Specifically, a persistence framework reads, writes, and deletes objects to/from permanent storage. The persistence provides full encapsulation of the persistence mechanism. Application developers can send a save, delete or retrieve message to the persistence framework and the framework will handle the rest of the interaction with the database.

The persistence framework also provides the ability to implement persistence behavior on multiple objects concurrently. The framework supports saving, deleting, or retrieving many objects at once depending upon a specific criterion.



The persistence layer can implement transactional behavior on objects. A transaction is defined as a combination of actions implemented on several objects concurrently. An example is adding an object to a database and deleting another object from another database and being able to rollback the entire transaction if an error occurred.

The persistence layer uses pooling resources available to help maintain efficient use of the database. If a single client has the ability to request every record from a datasource, then that client may be able to consume almost all resources of that datasource. The persistence framework uses a controlled approach that does not allow runaway use of a resource.

The persistence layer can dynamically run stored procedures on the database or submit SQL directly from the application. The persistence framework includes application supplied data classes that allow the framework to know the schema of the database it is connected to.

2.6 Search

The search framework simplifies, standardizes, and improves the use of the Autonomy search engine. This framework complies with J2EE standards instead of using CGI as in the current search engine interface. The framework consists of a search classes that provides a common way to access the Autonomy HTTP API and utilize its features.

The search wrapper implements the following Autonomy features:

- Query search engine
- Natural Language or "Fuzzy" query search engine
- Display search results
- Suggest additional search results



3 Detailed Designs

This section provides detailed designs for the ITA R2.0 RCS frameworks.

3.1 Component Factory

3.1.1 Introduction

The component factory framework, also known as the Object Factory, is a producer of objects. The component factory is implemented using an SFA-customized version of Accenture's GRNDS (General and Reusable Netcentric Delivery Solution) factory framework. The GRNDS code has been significantly modified to meet SFA application development requirements, and to work optimally in the SFA technical environment. The GRNDS code formed the basis for the framework; however, the SFA Component Factory is now the standard.

Additional technical documentation is provided in this document in order to help developers use the Component Factory framework in their applications. The framework has been modified to:

- Support applications running on the IBM WebSphere Application Server (WAS)
- Limit the framework to a local class since SFA applications are not currently implementing Enterprise JavaBeans
- Integrate with the RCS Exception Handling framework
- Integrate with the RCS Logging framework

3.1.2 System Overview

The component factory framework is created using the Java programming language. The component factory will run on IBM's WebSphere Application Server (WAS). However, there is not anything in the package that will limit the component factory to only run on WAS. The component factory encapsulates object creation logic by providing an instance of an object and not revealing its implementation.

The component factory implements the factory method pattern. There is no single class that makes the decision as to which subclass to instantiate. Instead, the superclass (SFAProducer) defers the decision to each subclass. This pattern does not actually have a decision point where one subclass is directly selected over another subclass. A program written using this pattern defines an abstract class that creates objects but lets each subclass decide which object to create.

The component factory framework defines a general and extensible factory mechanism. It allows developers to completely decouple how objects and components are instantiated from their use. In addition, the meta-information defines the object production mechanism, and provides surrounding context via configuration properties thereby allowing production to be changed



with minimal effort. As a result the component factory provides a very powerful service, and enables the definition of clear migration strategies from one architectural approach to another.

The component factory includes standard methods of producing objects when parameters are passed, common parameters to be passed when producing objects, and examples for using the component factory framework. Components can be registered by adding properties to the factory's defined configuration domain. The RCS component factory framework offers a standardized approach to retrieving system components through a predefined lookup mechanism.

Implementing the component factory will provide application teams with the following benefits:

Rapid development and code reuse

If an environment needs to be configured before creating an instance of an object then using the component factory will ease the object creation by allowing a developer to call the produce method of the factory and not to worry about setting up an environment. A properties file that is used to create the object will configure the environment. This will help promote rapid software development and code reuse.

Complex object creation

If object creation is complex (i.e., a class uses its subclasses to specify which objects it creates) then using the component factory can ease the development effort.

Migrating to different environments

If the production and development environments are not a mirror image of each other, then the developer can encapsulate all the configuration information in the SFAFactory class and just create an instance of an object. As a result, the component factory enables the definition of clear migration strategies from one architectural approach to another.

Technology change

The technology underlying the creation of an object can change. Future releases of RCS will extend the component factory to support EJBs and JDBC 2.0 DataSources. After the component factory is extended it can be used to create lite-EJBs that are collocated with servlets and JSPs. Later these lite-EJBs can be converted into complete EJBs and deployed onto an application server without changing any of the previously defined servlets and JSPs.

Arbitrary data types

The component factory can be used with arbitrary data types. The component factory is organized in such a way that it can instantiate other classes without being dependent on any of the classes it instantiates.

Adding new classes

The component factory can easily add new subclasses without impacting the existing classes.



Thus, as the application gets more complex (e.g., the application uses EJBs) then the developer can easily extend the component factory by additional classes that create different types of objects.

3.1.3 Design Considerations

3.1.3.1 Assumptions and Dependencies

It is assumed that the component factory framework will function in a J2EE application server environment. As the current production server for SFA is IBM's WAS v. 3.5, the framework will be compiled using its required JDK version 1.2.2. It will also work with the current JavaServer Pages (1.1), Java Servlet (2.2), Java Messaging Service (1.0.1), and Java Database Connectivity (2.0) specifications for this server.

3.1.3.2 Goals and Guidelines

The component factory framework will provide a robust framework that can easily be utilized by any SFA development team building applications using Java and WebSphere (although there is nothing in this package that ties it to WebSphere). The component factory framework is a producer of objects that accepts some information about how to create an object, such as a reference, and then returns an instance of that object. The components must be registered with the component factory.

3.1.3.3 Development Methods

This framework will be developed using general object-oriented software development techniques as specified in any standard text on the Java programming language. As the framework itself is fairly straightforward in the class and relationship patterns it employs, no object oriented modeling tool or methodology was specifically used in its design. However, the standard class and sequence diagrams are provided in this document in order to illustrate the factory's structure. These diagrams should assist developers who are unfamiliar with this framework.

3.1.4 System Architecture

The component factory framework has six classes. The class diagram shows how the classes are related to each other. Each class is described below.

3.1.4.1 SFAObject

The SFAObject extends Object. It is a base class for the component framework. It overrides some of the methods [e.g., equals(), hashCode(), toString() etc.] of the object class. The SFAProducer class extends this class.



3.1.4.2 SFAProducer

The SFAProducer is an abstract class. SFAProducer objects are used to retrieve components of a specific type. A local class represents the component. The SFAProducer class is a super class of the SFALocalProducer.

3.1.4.3 SFAFactory

The SFAFactory class provides an extensible mechanism to associate producers, targets, and context together to produce objects. The main responsibility of the SFAFactory is to retrieve a target using the proper producer. The task of producing the target is delegated to the SFAProducer. The SFAFactory contains the produce method that is typically the only method in the framework that developers call directly. The SFAFactory defines the strategy used to produce an object reference for a given key.

3.1.4.3.1 Producer

Producers define the strategy to produce an object reference for a given key. Producers are named entities that produce objects based on configuration properties in <DOMAIN>.properties file. Producers may be configured using the form indicated below.

```
sfa.factory.producer.<name>=<classname>  
sfa.factory.env.producer.<name>.<property>=<value>
```

The first property form is used to register a named producer. The second property form is used to set an environment property available for the producer across targets. This value can be overridden by a target specific property value.

3.1.4.3.2 Default Producer

A default producer may be defined by setting the property, sfa.factory.producer.default, to the desired producer class. If no default is set, then a producer is not used for the target and a warning should be logged. The RCS logging framework should be used for all logging purposes. A default producer is set through the properties as shown below. sfa.factory.producer.default = <classname>

Environment properties for the default producer are set through the properties as shown below.

```
sfa.factory.env.producer.default.<defaultproperty>=<value>
```

3.1.4.3.3 Target

Targets logically represent what clients want to create. Targets are set to be produced by a named producer. More specific properties may be applied to each target. Producers use these properties to produce target instances. Typically, target properties override more general



producer level properties with the same name, but this is dependent on the producer implementation (via the `SFAProducer.getProperty()` method).

Targets are configured as shown below.

```
sfa.factory.target.producer.<target-name>=<producer-name>  
sfa.factory.env.target.<target-name>.<property>=<value>
```

The first property form is used to register a target with a named producer. The second property form is used to set an environment property available to the producer for that target. This value may override a more general producer property value.

If a target is not explicitly set to be produced by a named producer, it will be assigned to the default producer. Target environment properties may be set even if the target is not explicitly registered with a producer. These properties will be used in conjunction with the default producer properties during production.

3.1.4.4 ProducerContainer

The `ProducerContainer` is an inner class of `SFAFactory`. This class defines the strategy used to produce an object reference for a given key. It calls the `produce()` methods for other classes (e.g., `SFALocalProducer`), which will return an instance of an object.

3.1.4.5 SFALocalProducer

The `SFALocalProducer` class extends `SFAProducer` class. This class uses simple reflection to create the target instance. This class uses the `classname` property to identify the class type producer for the target. This property is required, and must specify the fully-qualified Java classname for a class that can be found using the `Class.forName()` method.

Domain classes are a type of entity that may be registered with the `SFAFactory`. The class files for Domain classes are collocated with the `SFAFactory` client. As a result, to register them, the only thing that must be done is registering the class name with a logical name, and setting the target's `classname` property.

3.1.4.6 SFAFactoryException

The `SFAFactoryException` class throws exceptions while producing a reference to a component. This exception typically points to component factory misconfiguration. This class should be able to extend the RCS exception handling framework.



3.1.5 Detailed System Design

3.1.5.1 Component Overview

The SFAFactory provides an extensible mechanism to associate producers, targets and context together to produce objects.

Producers define the strategy used to produce an object reference by a given key. Objects are instantiated by a simple reflection-based technique. The SFAFactory defines a single facility that consolidates those strategies and enables their selection to be deferred or changed. A default producer may be defined by setting the property to the desired producer class. Targets logically represent what clients want to create. Targets are set to be produced by a named producer. Producers use these properties to produce target instances.

3.1.5.2 Component Definitions

SFAObject

Class Name:	SFAObject
Component:	Component Factory
Description:	This class defines behaviors which are required, suggested and useful for typical classes.
Package:	sfa.gov.ed.ita.componentfactory
Superclass:	Object

Attribute	Type	Description
Protected:		
serialVersionUID	static final long	Serial id

Con/Destructors	Arguments (Type, Name)	Description
SFAObject	None	Default constructor



Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Protected:			
doToString	None	String	This method is called from the template toString method. Subclasses implement doToString() to help produce an appropriate string representation of the object.
isEqualIdentityBased()	None	boolean	Return true if equals() should be based on the identity of the object (by its reference in memory), or false if equals() is based on a logical comparison.
Public:			
equals	Object rhs_	boolean	This template method implements the baseclass form of equals. All classes should implement equals. Subclasses should override equals(), however they should call super.equals().
hashCode	None	int	This method implements the canonical baseclass form of hashCode(). All classes should implement hashCode(). Subclasses should override hashCode(), however they should call super.hashCode() .
toString	None	String	This template method implements a canonical form of toString. Subclasses should override the doToString method, producing an appropriate string value.



SFAProducer

Class Name:	SFAProducer
Component:	Component Factory
Description:	This class defines the strategy to produce an object reference by a given key.
Package:	gov.sfa.ed.ita.componentfactory
Superclass:	SFAObject

Attribute	Type	Description
private:		
m_env	Properties	Property file to set the environment

Con/Destructors	Arguments (Type, Name)	Description
SFAProducer	None	Default constructor - initialize m_env.

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
equals	Object rhs_	boolean	Compare objects and property files.
getEnvironment	None	Properties	Get the environment (m_env). This is a final method.
getProperty	String name_ Properties ctx_	String	Return the corresponding property listed within ctx_ or producer environment set.



Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
hashCode	None	int	Return hash code.
produce	String name_ Object[] args_ Properties ctx_	Object throws SFAFactoryExcept ion	This is an abstract method. Used by clients to actively produce the object identified by the ctx_ argument. The ctx_ argument is provided to the producer to assist producing the return value.
setEnvironment	Properties env_	void	This is a final method. Used prior to executing a retrieval to set the configuration environment, if appropriate, that will be initialized into the retrieved component.



SFAFactory

Class Name:	SFAFactory
Component:	Component Factory
Description:	This class provides an extensible mechanism to associate producers, targets, and context together to produce objects.
Package:	sfa.gov.ed.ita.componentfactory
Superclass:	Object

Attribute	Type	Description
Public:		
DEFAULT_PRODUCER_NAME = "default";	static final String	Class constant
Private:		
REG_DEFAULT_PRODUCER = "sfa.factory.producer.default";	static final String	Class constant
ADD_DEFAULT_ENV_PROP = "sfa.factory.env.producer. default. ";	static final String	Class constant
REG_TARGET_PREFIX = "sfa.factory.target.producer. ";	static final String	Class constant
ADD_TARGET_ENV_PROP = "sfa.factory.env.target. ";	static final String	Class constant
REG_PRODUCER_PREFIX = "sfa.factory.producer. ";	static final String	Class constant
ADD_PRODUCER_ENV_PROP = "sfa.factory.env.producer. ";	static final String	Class constant
ms_defaultProducer	static SFAProducer	Default producer
ms_producers	static Hashtable	Hashtable that list producers
ms_targetProducers	static Hashtable	Target producer
ms_isInitialized	static boolean	Flag for initialization



Attribute	Type	Description
ms_configEnv	static Properties	Configuration file

Con/Destructors	Arguments (Type, Name)	Description
SFAFactory	None	Default constructor

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
addProducer	String name SFAProducer p_	void	This is a static synchronized method. Add a producer dynamically to the SFAFactory.
addTargetProducer	String target_ Properties ctx_	void	This is a synchronized method. Add a target, and its associated property context, to the SFAFactory. Calls addTargetProducer (string, string, properties) method.
addTargetProducer	String target_ String producerName_ Properties ctx	void	This is a synchronized method. Adds a target, and its associated property context, to the SFAFactory. The name of the producer responsible for target production must be specified.
Produce	String name_	Object Throws SFAFactoryException	Returns a reference to the component for the given name_. Calls produce (string, object) where object is null.
Produce	String name_ Object[] args_	Object Throws SFAFactoryException	Returns a reference to the component for the given name_.



Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
setEnvironment	Properties env_	void	Sets the configuration environment for the SFAFactory facility. This environment is used during the facility's initialization and refreshing processes. This method should set the factory's configuration environment. Another option would be to import grnds-core.jar file in the program that sets the configuration environment.
Private:			
doAddProducerProperty	String configProperty_ Hashtable envs_	void	This is a synchronized method. It configures SFAProducer property.
doAddTargetProperty	String configProperty_ Hashtable ctx_	void	This is a synchronized method. It configures target property.
doPopulateProductionEnvironments	Hashtable producerEnvs_	void	Populate producer context properties.
doPopulateTargetContexts	Hashtable ctx_	void	Populate target context properties.
doRegisterProducer	String name_ String property_	void throws SFAFactoryException	Register the producer.
doRegisterTarget	String target_ String property_	void throws SFAFactoryException	Register the target.
getDefaultEnvironment	None	Properties	Get the default environment.
getProducerContainer	String name_	ProducerContainer	This is a synchronized method. It gets the ProducerContainer..
Init	None	void throws SFAFactoryException	Configure environment for Component Factory and initialize the factory.
initProducers	None	void throws SFAFactoryException	Initialize the SFAFactory.



Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
initTargetProducers	None	void throws SFAFactoryExcept ion	Initialize the target producer.



ProducerContainer

Class Name:	ProducerContainer
Component:	Component Factory
Description:	This class defines the strategy to produce an object reference by a given key.
Package:	gov.sfa.ed.ita.componentfactory
Superclass:	SFAProducer

Attribute	Type	Description
private:		
m_producer	SFAProducer	Producer
m_ctx	Properties	Property file

Con/Destructors	Arguments (Type, Name)	Description
ProducerContainer	SFAProducer p_	Initialize all the private variables.

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
produce	String name_ Object[] args_	Object throws SFAFactoryExcept ion	This method calls the SFAProducer's produce method.
setContext	properties ctx_	void	Set the m_ctx variable.



SFALocalProducer

Class Name:	SFALocalProducer
Component:	Component Factory
Description:	This class uses the classname property to identify the class type producer for the target. It uses simple reflection to create the target instance.
Package:	gov.sfa.ed.ita.componentfactory
Superclass:	SFAProducer

Attribute	Type	Description
private:		
serialVersionUID	static final long	serial id

Con/Destructors	Arguments (Type, Name)	Description
SFALocalProducer	None	Default constructor - initialize m_env.

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
produce	String name_ Object[] args_ properties ctx_	Object throws SFAFactoryExcept ion	Create an instance of the local class.



SFAFactoryException

Class Name:	SFAFactoryException
Component:	Component Factory
Description:	This exception may be thrown while producing a reference to a component. This exception typically points to Component Factory misconfiguration
Package:	gov.sfa.ed.ita.componentfactory
Superclass:	SFAException

Attribute	Type	Description
private:		
serialVersionUID	static final long	Serial id

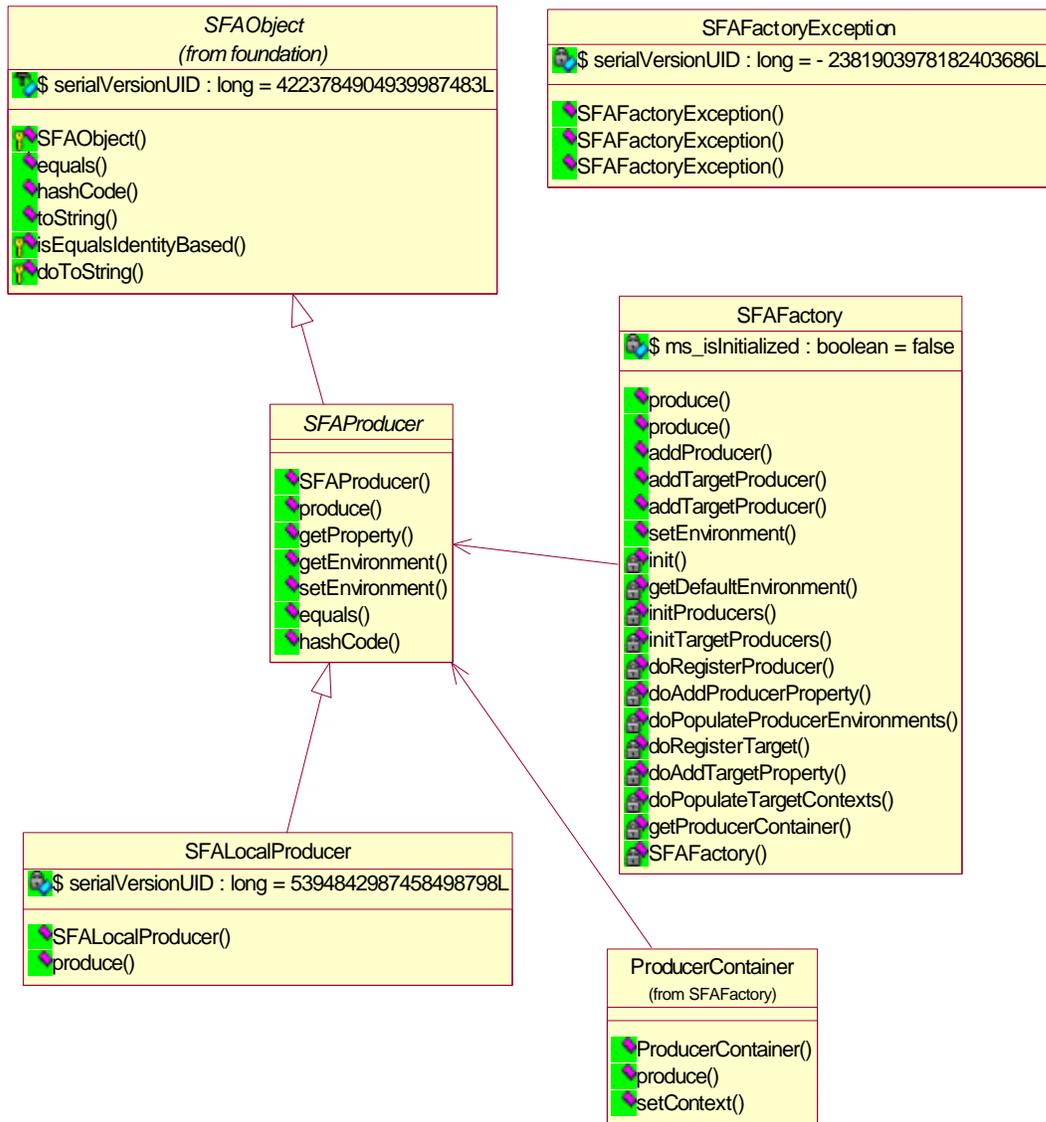
Con/Destructors	Arguments (Type, Name)	Description
SFAFactoryException	None	Default constructor
SFAFactoryException	String msg_ Throwable error_	Calls the super class's constructor.
SFAFactoryException	String msg_	Calls the super class's constructor.

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
None			



3.1.6 Class Diagrams

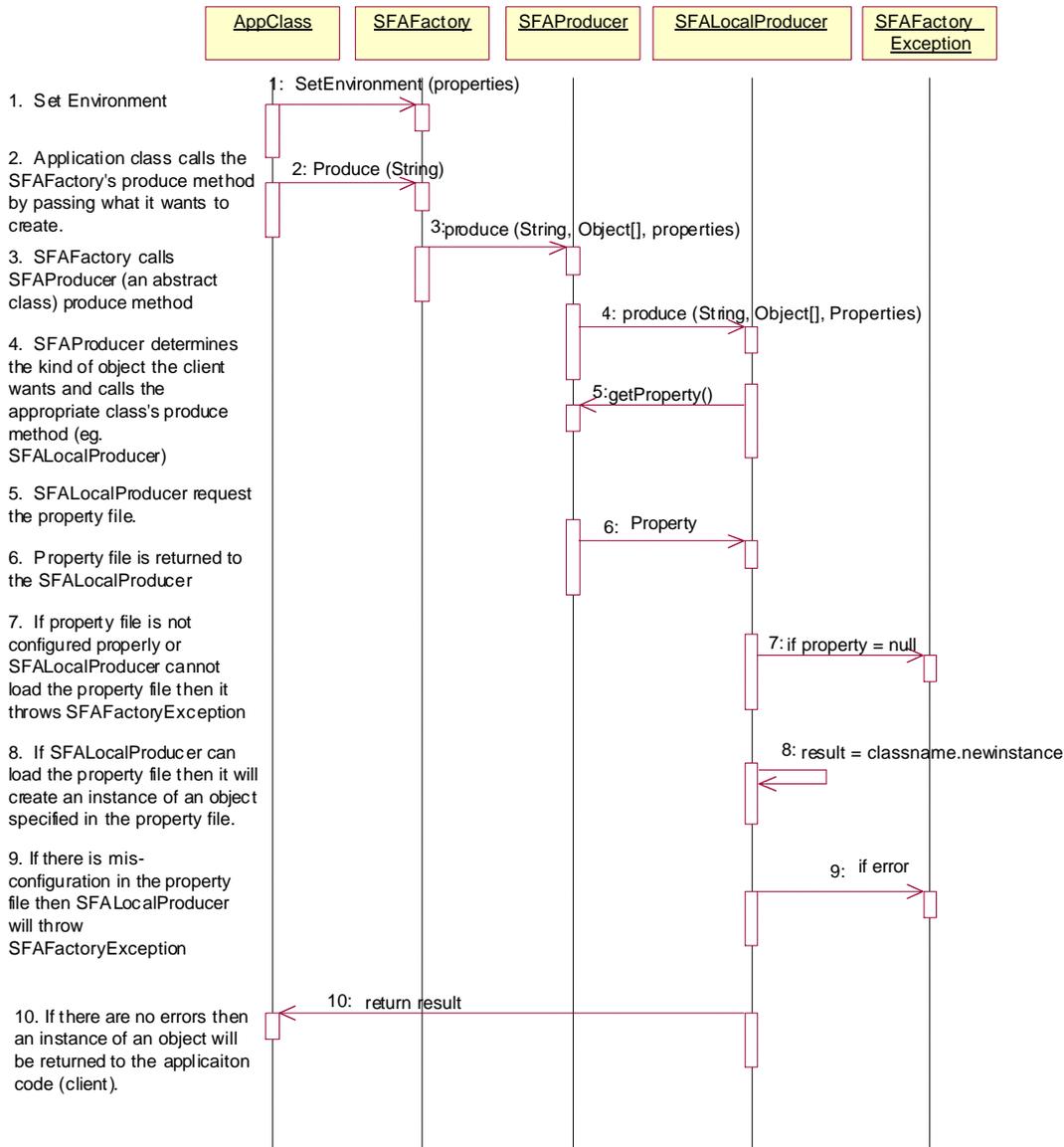
3.1.6.1 Component Factory Class





3.1.7 Interaction Diagram

This sequence diagram illustrates how the SFALocalProducer fulfills its object request. Understanding any one of the object calls, should serve as good basis for all other object request calls in this package.





3.1.8 References

- Design Patterns: Elements of Reusable Object Oriented Software, Gamma, Helm, Johnson, Vlissides. 1998
- GRNDS framework
<https://onesource.accenture.com>
- Sun Java web-site
<http://java.sun.com>



3.2 E-Mail

3.2.1 Introduction

The e-mail framework provides SFA with a common way to generate e-mail messages from applications. The e-mail framework uses Sun Microsystems' JavaMail API 1.2, a part of the J2EE standard, which provides a standard interface for Java programs to send e-mails to a Simple Mail Transport Protocol (SMTP) Mail server.

Previously, SFA applications used different methods to send e-mail messages. These methods included using a previous version of the JavaMail API and a propriety design that retrieves messages out of an Oracle database, processes each with UNIX shell scripts, and then sends the e-mail with the operating system utility sendmail.

3.2.2 System Overview

The e-mail framework will be implemented using a SFA customized version of Jnet's SMTP Client JavaBean. The e-mail framework uses Sun Microsystems's JavaMail API 1.2, which provides a standard interface for Java programs to send e-mails to a SMTP Mail server. The JavaMail API is in turn built upon the Java Activation Framework (JAF), allowing complex message submitting and retrieving through different protocols. These standard APIs provide a platform independent and protocol independent framework to build Java technology based mail and messaging applications.

The e-mail framework provides applications with the following features:

- Dynamically set all elements of an e-mail ("To" address, "From" address, subject, etc.)
- Send attachments to e-mail
- Set multiple e-mail addresses within the "To", "From", and "Reply To" fields
- Dynamically set the SMTP Server
- Verify e-mail parameters meet minimum standards for delivery
- Send a real time e-mail to a SMTP Server
- Process batch e-mails and send to a SMTP Server at a specified time

3.2.3 Design Considerations

3.2.3.1 Assumptions and Dependencies

It is assumed that this framework will function in a J2EE application server environment. As the current production server for SFA is IBM WebSphere 3.5.3, the framework will be compiled using its required JDK version 1.2.2. It should also work with the current JavaServer Pages (1.1), Java



Servlet (2.2), Java Messaging Service (1.0.1), and Java Database Connectivity (2.0) specifications for this server.

3.2.3.2 General Constraints

The e-mail framework will require that a SMTP Server is available so that e-mails generated by the client application may route mail to it. Currently, ITA applications use the operating system utility sendmail that is available on the majority of Unix machines.

3.2.3.3 Goals and Guidelines

The e-mail framework will provide SFA with a robust framework that can easily be utilized by SFA development teams building applications in Java, or more specifically WebSphere (although there is nothing in this package that ties it to WebSphere). In the past, the SFA operations group has reported limited success in tracing and debugging e-mail problems. The e-mail framework will be reliable and easy to debug if problems occur. To this end, the RCS logging and exception handling frameworks will be used to document any errors that the e-mail framework encounters.

Another goal of this framework is to standardize SFA applications on a single interface to e-mail. As specified before, different implementations of e-mail have occurred in previous SFA applications and it is important that a supportable, maintainable, and standard e-mail interface be used across all SFA applications. The RCS e-mail framework will handle real-time individual e-mails as well as large volumes of scheduled batch e-mails.

A final goal is to insure that the performance of this e-mail framework does not hinder the performance of the applications using it. Since the ITA environment SMTP Server is local, performance should not be an issue, but the ITA team will ensure adequate performance testing before releasing the e-mail framework.

3.2.3.4 Development Methods

This framework was developed using general object-oriented software development techniques as are specified in any standard text on the Java programming language. As the framework itself is fairly straightforward in its class and relationship patterns it employs, no object oriented modeling tool or methodology was specifically used in its design. However, the resulting class files have been documented with standard class diagrams and sequence diagrams using Rational Rose in order to illustrate its structure more readily. These diagrams should be helpful to programmers unfamiliar with this framework.

3.2.4 System Architecture

The e-mail Framework provides the ability to create, manipulate, and send e-mails from any Java application running within a J2EE Application Server. Developers can use the simplified e-mail framework to send e-mail or extend the JavaMail API's themselves for specific e-mail features.



3.2.4.1 Subsystem Architecture: SFA E-mail JavaBean

As specified in the System Overview, the e-mail framework uses a SFA customized version of the Jnet SMTP JavaBean to provide simplified interfaces to the Sun Microsystems's JavaMail API. This JavaBean provides public methods to create and manipulate e-mail via Java programs.

3.2.4.2 Subsystem Architecture: SMTP Server

The e-mail framework requires that the framework have access to a SMTP server. SMTP is the protocol that most Internet vendors implement to send e-mails across the Internet. The SMTP server is the workhorse that the e-mail framework will connect to and forward e-mails that SFA applications produce. Once the SMTP server has received the e-mail, it will connect to its forwarding partner (another SMTP server) and forward the e-mail to the Internet. Currently, the e-mail framework uses the SMTP Server that exists on the WebSphere servers.

3.2.4.3 Subsystem Architecture: JavaMail 1.2 API

The e-mail framework provides wrapper classes that are based on Sun Microsystems JavaMail 1.2 API. The JavaMail 1.2 API offers a clean, object-oriented framework of classes that model a theoretical mail system. The JavaMail 1.2 API can support a variety of protocols including SMTP, POP, IMAP, and MIME protocols. Any Java based program can programmatically send and receive e-mail through the JavaMail 1.2 API.

3.2.4.4 SubSystem Architecture: Java Activation Framework

The E-mail framework also depends on the Java Activation Framework. (JAF) The JavaMail API leverages the capabilities for dealing with complex data types from the JAF, which is part of the Glasgow JavaBeans specification. JAF provides Java with similar capabilities that plug-ins provide for web browsers. The JAF allows for the querying and handling of multi-media data types.

3.2.4.5 SubSystem Architecture: ITA Logging and Exception Handling Framework

The RCS logging and exception handling frameworks have been added to the e-mail framework to enhance debugging and tracing abilities. This will aid the SFA application and operations teams' ability to isolate problems that may occur within the framework.

3.2.5 Detailed System Design

3.2.5.1 SFASmtpClient JavaBean

The ITA E-mail framework includes a wrapper class that encapsulates the JavaMail API. This wrapper provides a simplified interface to the Sun JavaMail API and allows easy creation and manipulation of e-mail traffic. The SFASmtpClient JavaBean is aimed at supporting SMTP, which



is the predominate protocol for sending e-mail. Figure 1 below shows the interaction of a client program and a SMTP server using the ITA E-mail framework.

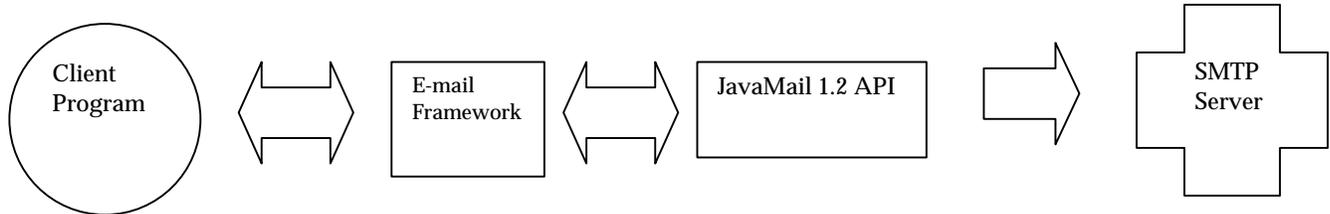


Figure 1: Interaction of Client Program and SMTP Server using ITA E-mail framework

Real Time versus Batch E-mail Processing

The SFASmtpClient JavaBean can be used in two different scenarios. The first scenario involves a user receiving an e-mail in real-time from a Web interface. An example of this is an SFA employee nominating another SFA employee via a Web page and immediately receiving confirmation e-mail after the completing the nomination form. Once the SFA employee submits the nomination form, the application server processes the information, builds an e-mail, and sends the e-mail immediately.

The second scenario involves an example like a monthly newsletter that is sent out to a group of users on a periodic basis. This scenario needs a scheduler agent that kicks off a command line Java program that interfaces with the RCS e-mail framework and builds batch sets of e-mails. This scenario uses the same e-mail framework as the real-time scenario, but is not processed through the application server. With this scenario a newsletter can be sent to hundreds of recipients on a scheduled basis.

Both solutions use the Java-based RCS e-mail framework and leverage the RCS logging and exception handling frameworks. It's important that a common interface is used between the command line batch and application server real time for support and maintenance reasons of e-mail capability.



3.2.5.2 Component Definitions

SFASmtpClient Class

Interface Name:	SFASmtpClient
Component:	e-mail
Description:	An e-mail class that verifies proper attributes is configured and simplifies the API interface before submitting e-mail to the Sun JavaMail API.
Package:	gov.ed.sfa.ita.email
Superclass:	None

Attribute	Type	Description
Private:		
hostname	String	holds the SMTP server hostname
HoldContent;	String	holds Text Content
HoldDebug	boolean	holds Debug Value
HoldFa	String[]	Holds array of names of Attached Files
HoldMp	MimeBodyPart[]	Holds MimeBody
HoldSentDate	Date	Holds current Date
HoldSubject	String	Holds Subject Line
HoldIAToAddress	javax.mail.internet.IneternetAddress[]	Holds an array of "To Addresses"
HoldIAFromAddress	javax.mail.internet.IneternetAddress	
HoldIARTA	javax.mail.internet.IneternetAddress[]	Holds an array of "Reply To Address's"

Con/Destructor	Arguments (Type, Name)	Description
SFASmtpClient	none	Constructor to build E-mail Session

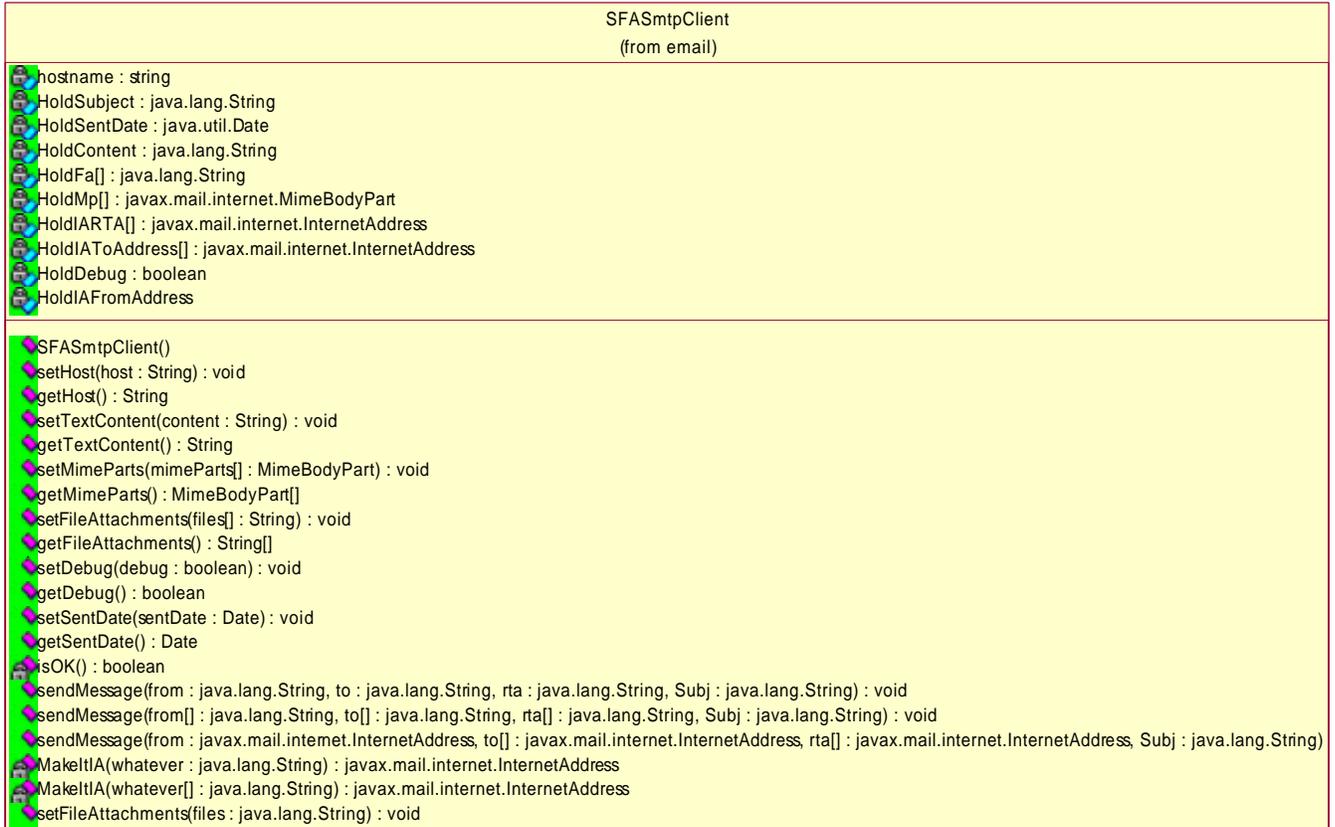
Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
setHost	string host	void	Set SMTP Hostname
getHost	none	string	Get SmtP Hostname
setTextContent	String content	void	Setter for TextContent
getTextContent	none	String	Getter for Text Content
setMimeParts	MimeBodyPart mimeParts[]	void	Setter for mimeparts(MimeBodyPart[])
getMimeParts	none	MimeBodyPart[]	Getter for mimeparts
setFileAttachments	String files[]	void	Setter for FileAttachment names(String [])
setFileAttachments	String filename	void	Setter for FileAttachment Names(String)



Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
getFileAttachments	none	String[]	getter for File Attachments
setDebug	boolean debug	void	Setter for trace mode
getDebug	none	boolean	Getter for trace mode
setSentDate	Date sentDate	void	Setter for SendDate
getSentDate	none	Date	Getter for Sent Date
isOk	none	boolean	Validation that required parameters are set
sendMessage	String from, String to, String rta, String Subj	MessagingException, AddressException	Method to interface with JavaMail to send message
sendMessage	String from[], String to[], String rta[], String Subj	MessagingException, AddressException	Method to interface with JavaMail to send message
sendMessage	String from, InetAddress to[], InetAddress rta[], String Subj	MessagingException, AddressException	Method to interface with JavaMail to send message
MakeItIA	String whatever	InetAddress	Utility method to translate String to a javax.mail.internet.InternetAddress



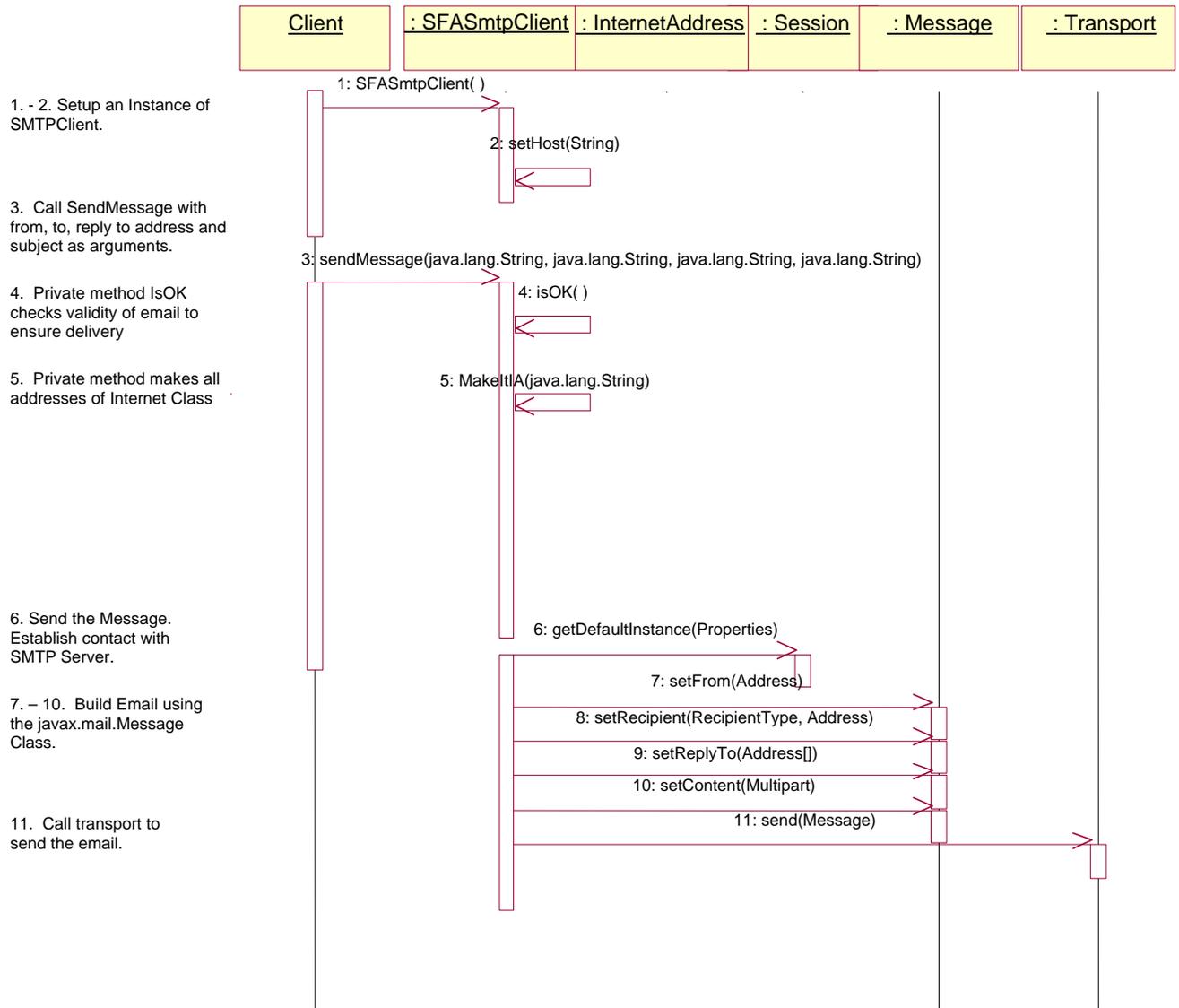
3.2.6 Class Diagrams





3.2.7 Interaction Diagrams

This sequence illustrates the interaction between a client object and SFASmtpClient class to build and send an e-mail to a SMTP Server





3.2.8 References

- Java Activation Framework JavaDocs online
<http://java.sun.com/products/javabeans/glasgow/javadocs>
- JavaMail 1.2 JavaDocs Online
<http://www.javasoft.com/products/javamail/1.2/docs/javadocs/index.html>
- Jnet JavaBean Store
<http://www.java-shop.com/jnet.htm>



3.3 Exception Handling

3.3.1 Introduction

The exception handling framework will allow all SFA WebSphere-based applications to share standard error handling and error logging procedures. The framework will also enable applications to throw a common, unified set of exceptions.

The purpose of exception handling is to catch problems in a program that would otherwise lead to program error and failure conditions. An error in Java is an unrecoverable abnormal condition. For example, an error can occur when a networking connection is unexpectedly cut or the JVM runs out of memory.

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. An example is passing a null value to a method that expects a valid String object reference. Exceptions can be prevented from becoming an error through correct programming.

The RCS exception handling framework exists to provide a mechanism by which client programmers may be able to trap exceptional conditions within their code. The benefits of having a framework to handle exceptions are to minimize duplicated effort and save time for programmers during both development and maintenance of the application.

3.3.2 System Overview

Despite Java's superior support for exception handling, most applications benefit significantly from the creation of their own application-specific exceptions and handlers. It is usually helpful to catch Java's exceptions, then create and throw application-specific exceptions. These custom exceptions can contain detailed messages or even nested exceptions, which can help simplify the debugging process. The RCS exception handling framework defines the base class for SFA exceptions, provides a factory to ensure exceptions are generated correctly, and allows for integration into the logging framework through a trace object.

The exception handling framework functions within the general exception handling framework that is built into the Java language. When an exception is thrown, the following steps must take place for it to be handled properly:

- Within a try-catch block, an exceptional condition is generated
- The Exception object is created and the current path of execution is interrupted
- The exception handling mechanism completes execution of the program
- The mechanism finds the appropriate handler for the given exception
- Information about the exception is read from the exception object



- Based on what is determined about the exception, the handler acts appropriately to either:
 - Remediate the exceptional condition and return the program to an operational state, or
 - Release resources used by the application and exit gracefully

With the exception handling framework, exceptions are handled as they traditionally are in Java, but they can be handled in a more consistent and robust manner, with detailed troubleshooting messages and advanced logging capability.

3.3.3 Design Considerations

3.3.3.1 Assumptions and Dependencies

It is assumed that this framework will function in a J2EE application server environment. As the current production server for SFA is IBM WebSphere 3.5, the framework will be compiled using its required JDK version 1.2.2. It should also work with the current JavaServer Pages (1.1), Java Servlet (2.2), Java Messaging Service (1.0.1), and Java Database Connectivity (2.0) specifications for this server.

3.3.3.2 Goals and Guidelines

The exception handling framework will provide a simple yet useful framework that can easily be utilized by any SFA development team building applications in Java, or more specifically WebSphere (although there is nothing in this package that ties it to WebSphere). The three things this framework is intended to provide is consistency in approach, standardization of error messages, and out-of-the-box integration with the logging framework.

Error handling is an important piece of any development effort, and a standardized framework should go a long way towards easing this coding burden on application programmers.

3.3.3.3 Development Methods

This framework was developed using general object-oriented software development techniques as are specified in any standard text on the Java programming language. As the framework itself is fairly straightforward in its class and relationship patterns it employs, no object oriented modeling tool or methodology was specifically used in its design. However, the resulting source code has been documented with standard class diagrams and sequence diagrams in order to illustrate its structure more readily. These diagrams should be of great help to programmers unfamiliar with this framework.



3.3.4 System Architecture

3.3.4.1 Overview

The exception handling framework enhances the ability of an application to define and trap errors that may arise as the application executes. The exception handling framework includes several key components:

- SFAExceptionFactory
- SFAException
- SFATrace

These components will be described individually in the detailed system design.

The exception handling framework provided by Java is very generic. It gives very generic messages to the developer, which makes it difficult to find the actual cause of the exception. Because of this, a custom framework for exception handling can be of great benefit to a development effort. With a custom exception handling framework, individual exceptions can be identified and handled on a case-by-case basis.

3.3.5 Detailed System Design

3.3.5.1 Component Overview

The exception handling framework consists of a set of classes that are designed to encapsulate the information pertaining to different errors that may occur within the system.

The Java Exception class is extended from the Throwable class, which is extended from the Object class. SFAException, in turn, is extended from the Exception class. A class diagram indicating the inheritance tree can be found in Section 6.

The SFAException class is the base exception, which can be customized on an application by application basis. This class has three private attributes: `uniqueId`, `errorCode`, and `arguments`. Each private attribute has corresponding public `set()` and `get()` methods.

The SFAExceptionFactory is the class used to create exceptions. It should work in any application without modifications. This class is designed to operate to work as a Singleton (i.e., there is only one instance of it per VM or at least per classloader). It contains some exception-centered utility methods, such as `getHostId()`, which can be used to uniquely identify an exception as belonging to a specific machine. It also has a generic `logException()` method that can write out exception information to `System.err`.



The SFATrace object is extended from SFAException and is used to mirror an exception with the additional feature of dumping its information to the logging framework for processing. This component greatly extends the reporting capabilities of the exception handling framework over standard exception handling.

The SFAException class contains a set of generic error codes to serve as a starting point. Each project should spend time early on in the development cycle to define a list of error codes encompassing all potential error conditions their application may encounter.

With the exception codes, a list of exception messages should be supplied in an errorMessages.properties resource file. The message number in the resource file corresponds to the message constant defined in the customized SFAException handling class.



3.3.5.2 Component Definitions

SFAException Class

Class Name:	SFAException
Component:	Exception
Description:	SFAException is the generic exception class.
Package:	gov.ed.sfa.ita.exception
Superclass:	java.lang.Exception

Attribute	Type	Description
Private:		
uniqueId	java.lang.String	Holds the uniqueId for the error.
errorCode	long	Holds the errorCode number.
arguments	java.lang.Object[]	Holds any arguments specified for the error.
Public:		
BAD_DATE_FORMAT	static long	Pre-defined sample error code.
BAD_PROPERTY_FORMAT	static long	Pre-defined sample error code.
DATA_ACCESS_EXCEPTION	static long	Pre-defined sample error code.
EJB_HOME_TYPE_MISMATCH	static long	Pre-defined sample error code.
ERROR_CREATING_BEAN	static long	Pre-defined sample error code.
ERROR_LOADING_PROPERTY_FILE	static long	Pre-defined sample error code.
ERROR_OPENING_QUEUE	static long	Pre-defined sample error code.
ERROR_READING_QUEUE_MSG	static long	Pre-defined sample error code.
ERROR_SENDING_QUEUE_MSG	static long	Pre-defined sample error code.
GENERIC_SERVLET_ERROR	static long	Pre-defined sample error code.



Attribute	Type	Description
INSTANTIATION_EXCEPTION	static long	Pre-defined sample error code.
INVALID_BUSINESS_REQUEST	static long	Pre-defined sample error code.
INVALID_FIELD_CONTENTS	static long	Pre-defined sample error code.
INVALID_HOME_INTERFACE	static long	Pre-defined sample error code.
INVALID_MSG_CLASS	static long	Pre-defined sample error code.
JNDI_INIT_ERROR	static long	Pre-defined sample error code.
JNDI_LOOKUP_ERROR	static long	Pre-defined sample error code.
METHOD_NOT_IMPLEMENTED	static long	Pre-defined sample error code.
MISSING_CLASS	static long	Pre-defined sample error code.
MISSING_CREATE_METHOD	static long	Pre-defined sample error code.
MISSING_PROPERTY	static long	Pre-defined sample error code.
NEW_INSTANCE_FAILED	static long	Pre-defined sample error code.
QUEUE_COMMUNICATION_ERROR	static long	Pre-defined sample error code.
RECEIVED_ERROR_REPLY	static long	Pre-defined sample error code.
SQL_EXCEPTION	static long	Pre-defined sample error code.
TIMEOUT	static long	Pre-defined sample error code.
UNEXPECTED_EXCEPTION	static long	Pre-defined sample error code.
UNKNOWN_MSG_CLASS	static long	Pre-defined sample error code.
UNKNOWN_MSG_TYPE	static long	Pre-defined sample error code.
WRONG_ARGUMENT_TYPE	static long	Pre-defined sample error code.

Con/Destructor	Arguments	Description
	(Type, Name)	



Con/Destructor	Arguments (Type, Name)	Description
SFAException		Generic constructor.
SFAException	java.lang.String s	Constructor that sets a specified error message.

Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
getArguments		java.lang.Object[]	Gets the arguments for this Exception.
getErrorCode		long	Gets the error code for this Exception.
getUniqueId		java.lang.String	Gets the unique id for this Exception.
setArguments	java.lang.Object[]	void	Sets the arguments for this Exception.
setErrorCode	long	void	Sets the error code for this Exception.
setUniqueId	java.lang.String	void	Sets the unique id for this Exception.



SFAExceptionFactory Class

Class Name:	SFAExceptionFactory
Component:	Exception
Description:	The SFAExceptionFactory class is used to create SFAException objects.
Package:	gov.ed.sfa.ita.exception
Superclass:	None

Attribute	Type	Description
Private:		
instance	SFAExceptionFactory	Internal reference to itself - class is a singleton.
counter	int	Counter to generate unique ids.
uniqueFactoryCode	long	Stores the system time it was created as part of the unique id.
hostId	java.lang.String	Stores the host id.
errorMessages	java.util.ResourceBundle	The error messages file is loaded into this object.
Public:		
dateFormat	java.text.SimpleDateFormat	Stores format for dates.

Con/Destructor	Arguments	Description
	(Type, Name)	
SFAExceptionFactory		Generic Constructor.

Method	Arguments	Valid Responses	Description
	(Type, Name)	(Return Type, Exceptions Thrown)	



Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private:			
generateUniqueId		java.lang.String	Generates a unique id for the exception.
Public:			
createException	java.lang.Class classDef, long errorCode, java.lang.Object[] arguments, java.lang.Throwable previousException, java.lang.String className, java.lang.String methodName, java.lang.String additionalInfo	SFAException	Generates an SFAException object.
getHostId		java.lang.String	
getInstance()		SFAExceptionFactory	Accessor for the singleton object.
getMessage	SFAException exception	java.lang.String	Gets the error message associated with the exception.
handleUnexpectedException	java.lang.Throwable previousException, java.lang.String className, java.lang.String methodName, java.lang.String additionalInfo	SFAException	Returns an exception associated with a caught Throwable object.
logException	SFAException exception, java.lang.Throwable previousException, java.lang.String className, java.lang.String methodName, java.lang.String additionalInfo	void	Logs the exception, writing an XML string to standard error.



SFATrace Class

Class Name:	SFATrace
Component:	Exception
Description:	The SFATrace class provides interoperability with the ITA Logging Framework.
Package:	gov.ed.sfa.ita.exception
Superclass:	SFAException

Attribute	Type	Description
Private:		
none		
Protected:		
none		
Public:		
none		

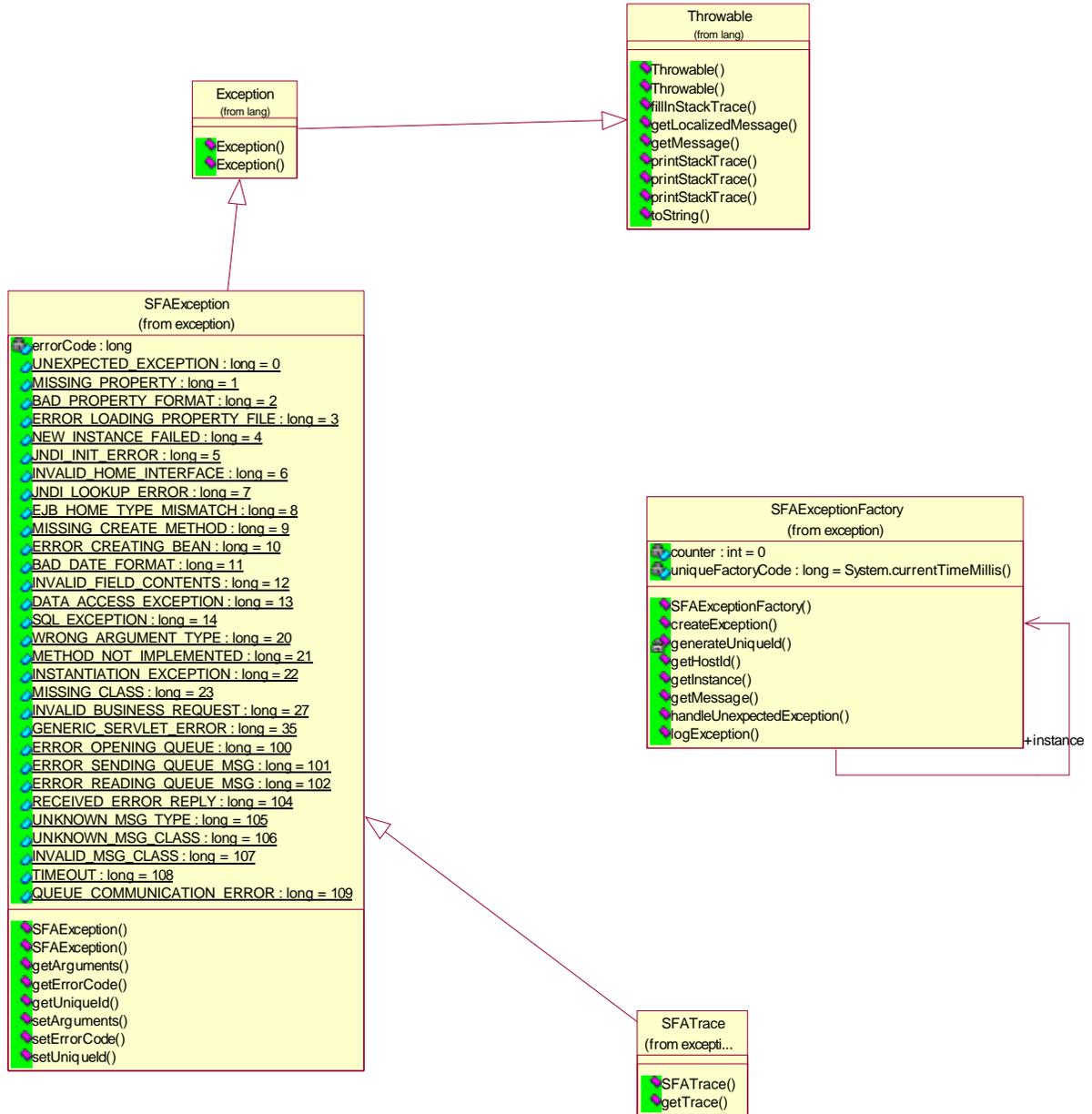
Con/Destructor	Arguments	Description
	(Type, Name)	
SFATrace	java.lang.Object obj, java.lang.String msg	Constructor for the trace object, which takes an SFAException and additional error message text.



Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private:			
none			
Protected:			
none			
Public:			
getTrace		java.lang.String	Gets a formatted trace message.



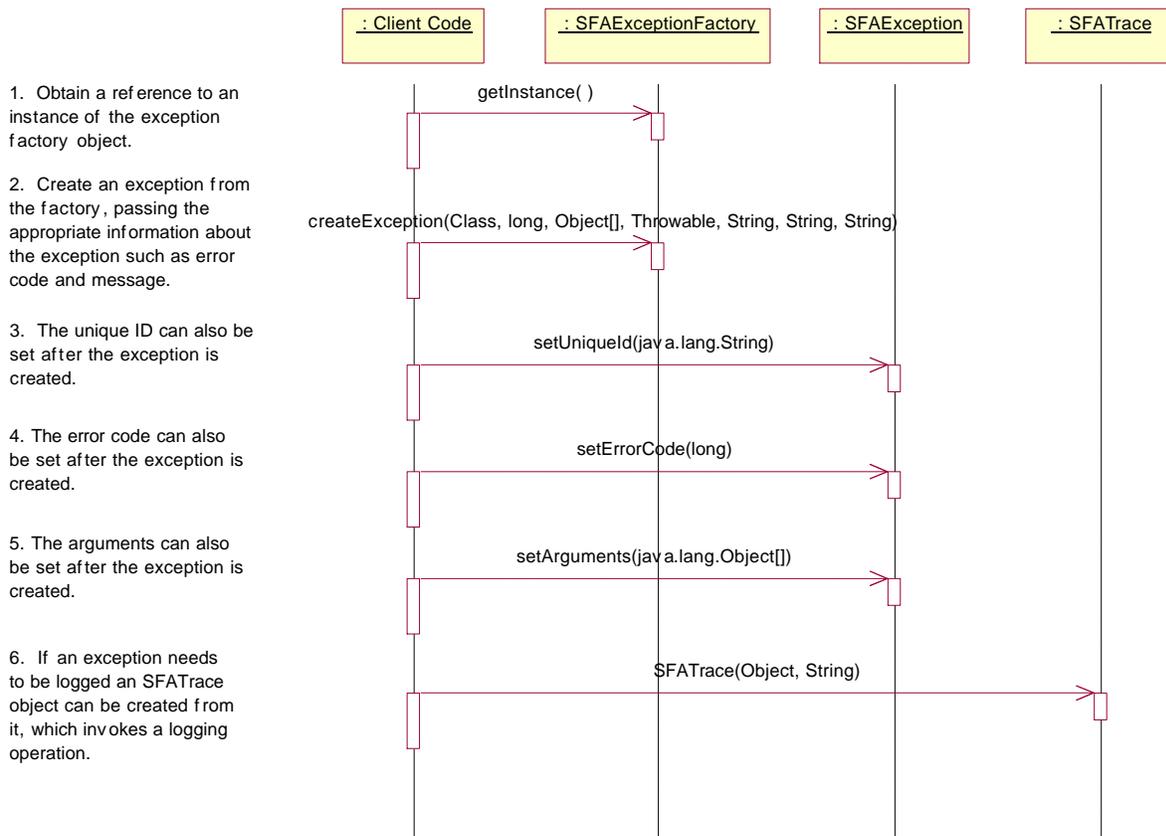
3.3.6 Class Diagrams





3.3.7 Interaction Diagrams

This sequence diagram illustrates the main object interactions involved in basic exception handling from client code. The client code must first obtain a reference to an instance of the SFAExceptionFactory. It then uses the factory to create SFAExceptions as needed, identifying the id, error code, and arguments (if any). Should the error need to be logged, it can be passed to the constructor of an SFATrace object, where logging is done as soon as the object is instantiated.





3.3.8 References

- Java Tutorial on exception handling
<http://java.sun.com/docs/books/tutorial/java/nutsandbolts/exception.html>
- JavaDoc on logging framework
<http://protomatter.sourceforge.net/latest/javadoc/com/protomatter/syslog/package-summary.html>



3.4 Logging

3.4.1 Introduction

The ITA custom logging feature set is implemented using an SFA-customized version of the Protomatter logging toolkit called Syslog. This toolkit is provided free for distribution through the Open Source Software group, is licensed under the GNU Library General Public License Version 2, and is free for both commercial and non-commercial use. Specific terms of the license are available at <http://www.gnu.org/copyleft/lgpl.html>. While the code has only been minimally modified from its initial state, it is hoped that the additional technical documentation provided in this document will serve to aid developers should they need to troubleshoot or modify this framework as it is implemented in SFA projects.

3.4.2 System Overview

The logging framework enhances the current logging ability of the ITA Web application servers (specifically WebSphere), by allowing programmers to dynamically set logging and tracing functionality without modifying tested source code. The logging framework also allows SFA to develop and enforce log formatting standards to ensure proper information is gathered if a failure occurs.

Syslog is a simple and robust logging system that is not tied to any specific application server. Syslog provides the following features:

- Simple logging API
- Background logging (*Configurable*)
- Multiple message severities
- Logs the name of the thread that issued the message (*Configurable*)
- Logs the name of the host that issued the message (*Configurable*)
- Arbitrary log channel names
- Pluggable log message listeners
- Pluggable log formatting modules for each listener
- Pluggable log policy modules for each listener
- Configuration can be modified on-the-fly while the system is running
- A running configuration can be written to XML for re-load later

Syslog lets the programmer log messages easily through a simple API. During development and testing, the messages may be simply sent to the console where the application is running or to a single log file. When the system is moved into production, log messages can be split up by



severity (for example, fatal messages may trigger the paging of an operations support resource) and log files may be rotated every night and archived. These kinds of configuration changes do not require changes to the code and can even be made while the system is running.

3.4.3 Design Considerations

3.4.3.1 Assumptions and Dependencies

It is assumed that this framework will function in a J2EE application server environment. As the current production server for SFA is IBM WebSphere v. 3.5, the framework will be compiled using its required JDK version 1.2.2. It should also work with the current JavaServer Pages (1.1), Java Servlet (2.2), Java Messaging Service (1.0.1), and Java Database Connectivity (2.0) specifications for this server.

3.4.3.2 Goals and Guidelines

The goal of this development was to provide a simple yet robust logging framework that could easily be utilized by any SFA development team building applications in Java, or more specifically WebSphere (although there is nothing in this package that ties it to WebSphere). An important part of any project, logging can be used as a debugging tool during development, and a troubleshooting tool once a system has been deployed in a production environment. Because most developers only implement logging as time permits, it is often implemented haphazardly. However, logging provides a way to see what is happening, good or bad, inside a running system. As such, it should be addressed with care and forethought rather than as a last minute burden.

3.4.3.3 Development Methods

This framework was developed using general object-oriented software development techniques as are specified in any standard text on the Java programming language. As the framework itself is fairly straightforward in its class and relationship patterns it employs, no object oriented modeling tool or methodology was specifically used in its design. However, the resulting source code has been documented with standard class diagrams and sequence diagrams in order to illustrate its structure more readily. These diagrams should be helpful to programmers unfamiliar with this framework.

3.4.4 System Architecture

3.4.4.1 Overview

The logging framework enhances the ability of an application to support personal or developer messages that can determine if a problem exists within an application. The logging framework includes the following key components:

- Loggers



- Filters
- Formatters
- Policy
- Channels

These objects can be configured in several ways to efficiently direct log output to multiple destinations with differing content and format. The logging framework can be configured programmatically or via some configuration file allowing support personal to enhance a logger's visibility by just changing a configuration file.

3.4.4.2 Subsystem Architecture: Loggers

Loggers listen for log messages from applications and provide methods that allow programmers to log messages via a simple API. Loggers have attributes that determine how a message is formatted, whether a message has visibility to the entire system or just one file, as well as whether a message is immediately flushed or buffered. Loggers represent the heart of the ITA logging framework and should be considered when detailing an application design.

3.4.4.3 Subsystem Architecture: Filters

Each logger in the Logging Framework has a log mask and each message has a severity. A masking of these two values determines whether the Logger allows the message to continue to the destination. The severities allowed within a message are:

- **DEBUG** - These are debugging messages usually placed by the programmers for the tracing and debugging purposes.
- **INFO** - These are useful informational messages about what is occurring.
- **WARNING** - These messages warn that something abnormal has happened, but that the system will attempt to recover from it. These messages are usually used by programmers to show that something is starting to go wrong.
- **ERROR** - These messages state that something abnormal has occurred, but that it is not severe enough to cause the system to fail in general. A specific task may fail and some users may get an error, but the system will keep going. Exceptions are generally logged at this level.
- **FATAL** - These messages inform that a fatal event has occurred. The system is probably now in a non functioning state. Someone should be paged.



It is important to actually use the different severity levels where they are appropriate. For example, if every message in the system is logged at the "debug" level, the operations team will be unable to judge the severity of the message.

Example 1

For example, if a Loggers log mask is set to INFO, then any message that comes in with a severity that is below INFO will be sent on to the destination. A message that has severity DEBUG will be ignored. With this Log Mask and Logger configuration, all info, warning, error, and fatal messages will show up at the destination. This is illustrated in Figure 2 below.

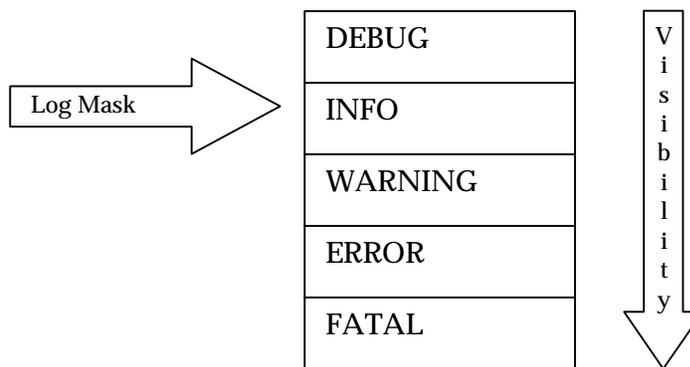


Figure 2: Example 1

Example 2

If the Log Mask is set to Fatal, like in Figure 3 below, then only Fatal messages show up at the destination.

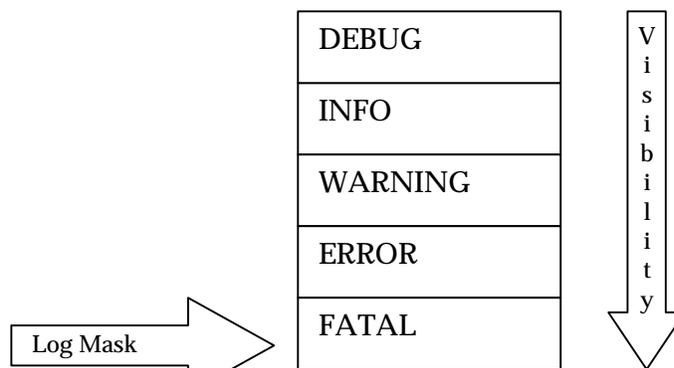


Figure 3: Example 2



Example 3

This concept can be extended so that two or more loggers can be used to send info and below messages to the console but fatal messages go to a log file. This is illustrated in Figure 4 below.



Figure 4: Example 3

3.4.4.4 Subsystem Architecture: Formatters

Formatters ensure that data submitted to Loggers will follow certain logging standards to ensure complete information. Formatters can be set to ensure that the current time and date are part of the log message. They also can specify the threadname, hostname, and time zone. An illustration of this is found in Figure 5 below.

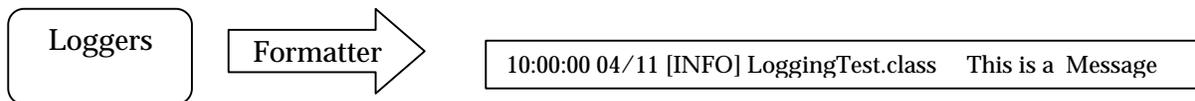


Figure 5: Formatters

3.4.4.5 Subsystem Architecture: Policy

Policies are rules combinations that developers can build using the ITA Logging framework. Application developers can develop several standard log policies that may encompass filters, severity and format requirements. Once the policies are developed then these policies are attached to the loggers.



3.4.4.6 Subsystem Architecture: Channels

The ITA Logging framework supports Log Channels. The idea is that some messages might need to be sent to multiple loggers. For example if a Java Application server supports multiple applications and each application has its own logger, then each logger might listen to a common channel for system wide messages. By default each logger listens to the channel “ALL_CHANNELS” and if a channel isn’t specified then a logger will listen to Channel “DEFAULT_CHANNEL”. An illustration of this is found in Figure 6 below.

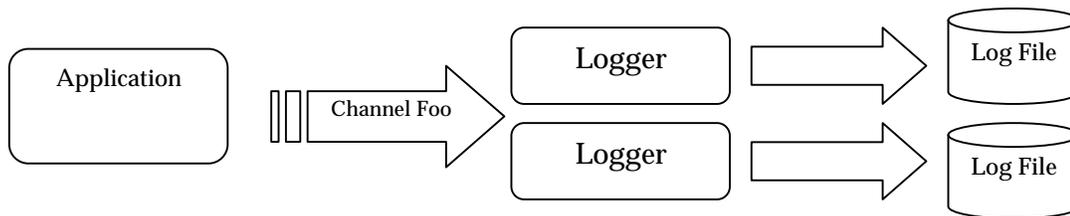


Figure 6: Channels

3.4.4.7 Subsystem Architecture: Configuration Mechanism

The RCS logging framework loggers can be set up within the application via API calls to the framework or read from a XML based document. The XML based document is a much better choice due to the fact that it permits changing the logger attributes by changing the XML document. The XML document will be read upon startup of the framework and initiate the loggers. If a change needs to occur to a logger, then the XML document is updated and the framework reinitialized.

3.4.4.8 Subsystem Architecture: Message Catalog

The logging framework can include a message catalog that will contain standard text messages that are expected to be used by the application. This message catalog is actually backed by a text properties file and read into the framework using the java.util.ResourceBundle class. This allows a developer to standardize messages and use a simple variable to substitute for complex text messages. The properties file uses a format of key=value

Contents of an errormessage.properties file example

```
PING_TIMEOUT="THE SYSTEM IS NOT RESPONDING TO A PING"  
MISSING_CLASS="A CLASS IS MISSING"
```



3.4.4.9 Subsystem Architecture: Smart Trace (Debug) Object

The logging framework includes a smart Debug class that produces a stack trace if the logger is enabled for it.

3.4.5 Detailed System Design

3.4.5.1 Component Overview

The Syslog class is a singleton (i.e. there is only one instance of it per VM or at least per classloader). It contains all configuration information related to where messages are routed, who is listening for those messages, etc. If it has been configured to log messages in the background (default) then Syslog maintains a slave thread that performs the actual logging. This is done so that calls to any of the logging methods return as quickly as possible and any extra work (writing to files) is done asynchronously.

Syslog maintains a list of loggers. Loggers listen to log messages being submitted. Each of these loggers is described in the next section of this document. These loggers implement the Syslogger interface -- mainly the `log(SyslogMessage message)` method that is called every time a message is sent to Syslog. Once the `log()` method is called on the Syslogger it decides if the message should be written to a file or console etc.

The Syslog class allows a program to log messages and objects at different severity levels in a standardized way. The implementation of the log is specified by an object that implements the Syslogger interface. There can be multiple log implementations, and each can have its own log mask or can inherit its log mask from Syslog (this is the default behavior). There are five severity levels: DEBUG, INFO, WARNING, ERROR, FATAL. They correspond to the numbers (constants) 0, 4, 8, 12, and 16, respectively. Logging can be enabled for any combination of these levels. The default setting for log is only WARNING and above.

There are several loggers included with the Syslog distribution. Each of these loggers extends BasicLogger and supports pluggable log policies. All of them except for the DatabaseLog support pluggable text formatting modules. These loggers are described below:

- **PrintWriterLog:** A logger that is attached to an instance of `java.io.PrintWriter`. This logger is generally attached to either `System.out` or `System.err` at runtime.
- **TimeRolloverLog:** A logger that writes to a file that is rotated every minute, hour, day or month.
- **LengthRolloverLog:** A logger that writes to a file that is rotated before it reaches a certain length.
- **FileLog:** A logger that simply writes to a file.



- **OpenFileLog:** A logger that writes to a file. The file is opened before and closed after each message is written. This is a very slow logger and should not be used unless there is a reason that justifies its use.
- **DatabaseLog:** A logger that writes messages to a table in a database. This is useful in situations where there are multiple machines involved in a project and there is need to have a unified view of everything happening on all the machines.

The default implementation of the Syslogger interface is the BasicLogger abstract class. This is the base class for all the loggers that are included with Syslog. The BasicLogger delegates the decision about paying attention to a given message to an implementation of the LogPolicy interface. The default implementation of the LogPolicy interface is the SimpleLogPolicy class. This policy knows about log message severity levels and about log. The BasicLogger also delegates its message formatting duties to an implementation of the SyslogTextFormatter interface. The default implementation of this interface is the SimpleSyslogTextFormatter class, which can be configured to format log messages in various ways.

If Syslog needs to be configured from inside another server or application, the Syslog.configure() method can be used. It will configure Syslog from an XML file. Syslog can also be configured programmatically.



3.4.5.2 Component Definitions

Component Name: JMSConstants
Classification: Java Public Interface
Definition: Constants for JMS-related Syslog functions.
Uses/Interactions: Implemented by JMSLog
Interfaces/Exports:

Data	
<code>static java.lang.String</code>	JMS_PROP_CHANNEL The message property of the message channel.
<code>static java.lang.String</code>	JMS_PROP_HOST The message property of the originating host's IP address.
<code>static java.lang.String</code>	JMS_PROP_LEVEL The message property of the message's severity level.
<code>static java.lang.String</code>	JMS_PROP_LOGGER The message property of the message logger's class name.
<code>static java.lang.String</code>	JMS_PROP_MESSAGE The message property of the message's short text.
<code>static java.lang.String</code>	JMS_PROP_MSG_TYPE The message property declaring that the given JMS message is a syslog message.
<code>static java.lang.String</code>	JMS_PROP_MSG_TYPE_VALUE The value of the message property declaring that the given JMS message is a syslog message.
<code>static java.lang.String</code>	JMS_PROP_THREAD The message property of the originating thread's name.
<code>static java.lang.String</code>	JMS_PROP_TIME The message property of the message send time.



Component Name: LogPolicy
Classification: Java Public Interface
Definition: The interface for the pluggable log policy system.
Uses/Interactions: Extends XMLConfigurable
Implemented by SimpleLogPolicy
Interfaces/Exports:

Method Summary
boolean shouldLog (SyslogMessage message) Determine if a log message should be logged given the information.
Methods inherited from interface com.protomatter.xml.XMLConfigurable
configure , getConfiguration



Component Name: RemoteLogReceiver
Classification: Java Public Interface
Definition: An interface for loggers that are receiving messages from a remote machine via RMI.
Uses/Interactions: Extends java.rmi.Remote
Implemented by RemoteLogReceiverImpl
Interfaces/Exports:

Method Summary
<pre>void log(java.lang.String ipAddress, java.lang.String loggerClass, java.lang.String channel, java.lang.String message, java.lang.Object detail, int level, java.lang.String threadName, long messageSendTime) Log callback method.</pre>



Component Name: SyslogChannelAware
Classification: Java Public Interface
Definition: An interface for objects that are aware of channels in syslog.
Uses/Interactions: Implemented by JdbcConnectionPool
Implemented by JdbcConnectionPoolConnection
Implemented by ProtoEJB

Interfaces/Exports:

Method Summary	
<code>java.lang.Object</code>	<code>getSyslogChannel</code> () This method should return the channel that messages coming from this object should be logged to if the call to Syslog didn't include a channel (or the channel specified was <code>null</code>).



Component Name: Syslogger
Classification: Java Public Interface
Definition: An interface for objects that will log things using the Syslog facility.
Uses/Interactions: Extends XMLConfigurable
 Implemented by BasicLogger

Interfaces/Exports:

Method Summary
java.lang.String getName() Get the name of this logger.
LogPolicy getPolicy() Get the log policy object used by this logger.
SyslogTextFormatter getTextFormatter() Get the log formatter object used by this logger.
void log(SyslogMessage message) Log an entry to the log.
boolean mightLog(java.lang.Object logger, int level, java.lang.String channel) Determine if it's likely that a message at the given level on the given channel(s) will be logged.
void setName(java.lang.String name) Set the name of this logger.
void setPolicy(LogPolicy policy) Set the log policy object used by this logger.
void setTextFormatter(SyslogTextFormatter formatter) Set the log formatter object used by this logger.
void shutdown() Shutdown this logger.
Methods inherited from interface com.protomatter.xml.XMLConfigurable
configure , getConfiguration



Component Name: SyslogMailSubjectFormatter
Classification: Java Public Interface
Definition: The interface for the pluggable message formatting system. It is used to format message subjects for the MailLog logger.
Uses/Interactions: Extends XMLConfigurable
Implemented by SimpleSyslogMailSubjectFormatter
Interfaces/Exports:

Method Summary
java.lang.String formatMessageSubject (SyslogMessage message) Format the message subject given the log entry.
Methods inherited from interface com.protomatter.xml.XMLConfigurable
configure , getConfiguration



Component Name: SyslogTextFormatter
Classification: Java Public Interface
Definition: The interface for the pluggable text formatting system. It is used to format log entries by subclasses of the BasicLogger logger.
Uses/Interactions: Extends XMLConfigurable
 Implemented by SimpleSyslogTextFormatter
Interfaces/Exports:

Method Summary	
void	formatLogEntry (java.lang.StringBuffer b, SyslogMessage message) Format the given log entry.
void	formatMessageDetail (java.lang.StringBuffer b, SyslogMessage message) Format the given log message's detail.
java.lang.String	getLogFooter () Get the footer text that should be included at the bottom of each log file.
java.lang.String	getLogHeader () Get the header text that should be included at the top of each log file.
void	resetDateFormat () Reset the formatter's date format.
Methods inherited from interface com.protomatter.xml.XMLConfigurable	
	configure , getConfiguration



Component Name: BasicLogger
Classification: Java Public Class
Definition: The base class for Syslogger implementations. This class provides common functions for setting the date format for log entries and for formatting dates. The default policy used by this logger is the SimpleLogPolicy policy. The default text formatter is the SimpleSyslogTextFormatter formatter.
Uses/Interactions: Implements the Syslogger interface
 Extended by DatabaseLog
 Extended by FileLog
 Extended by JMSLog
 Extended by LengthRolloverLog
 Extended by MailLog
 Extended by OpenFileLog
 Extended by PrintWriterLog
 Extended by RemoteLog
 Extended by TimeRolloverLog

Interfaces/Exports:

Field Summary
protected SyslogTextFormatter formatter
protected LogPolicy policy
Constructor Summary
BasicLogger () The default constructor -- configure() will need to be called.
Method Summary
protected void configure () Initialize with the default configuration.
void configure (org.jdom.Element e) Configure this logger given the XML element.
protected void formatLogEntry (java.lang.StringBuffer b, SyslogMessage message) Format the log entry using the current log formatter.
org.jdom.Element getConfiguration (org.jdom.Element element) Get the current configuration of this logger.
java.lang.String getName () Get this logger's name.
LogPolicy getPolicy () Get the log policy object used by this logger.
SyslogTextFormatter getTextFormatter () Get the log formatter object used by this logger.
boolean mightLog (java.lang.Object logger, int level, java.lang.String channel) Determine if it's likely that a message from the given logger at the given



level on the given channel will be paid attention to.
protected void resetDateFormat() Reset the text formatter's date format.
void setName (java.lang.String name) Set this logger's name.
void setPolicy (LogPolicy policy) Set the log policy object used by this logger.
void setTextFormatter (SyslogTextFormatter formatter) Set the log formatter object used by this logger.
protected boolean shouldLog (SyslogMessage message) A utility method to see if the current log policy says we should pay attention to this message.
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Methods inherited from interface com.protomatter.syslog.Syslogger
log , shutdown



Component Name: DatabaseLog
Classification: Java Public Class
Definition: A logger that writes to a database. See system JavaDocs for proper database/configuration file setup needed to use this logger.
Uses/Interactions: Extends BasicLogger
 Implements Syslogger (indirect)
 Implements XMLConfigurable (indirect)

Interfaces/Exports:

Fields inherited from class com.protomatter.syslog.BasicLogger	
	formatter , policy
Constructor Summary	
	DatabaseLog () You will need to call the configure() method if you use this constructor.
Method Summary	
	void configure (org.jdom.Element e) Configure this logger given the XML element.
org.jdom.Element	getConfiguration (org.jdom.Element element) Get the current configuration of this logger.
java.lang.String	getDriver () Get the classname of the driver.
int	getNumRetries () Get the number of retries before failure.
java.util.Properties	getProperties () Get the JDBC connection properties.
java.lang.String	getTablePrefix () Get the table prefix.
java.lang.String	getURL () Get the JDBC URL.
	void InitDatabase () (re)initialize the connection to the database.
	void log (SyslogMessage message) Log an entry to the log.
protected void	setDriver (java.lang.String driver) Set the driver to use.
protected void	setNumRetries (int retries) Set the number of retries before failure.
protected void	setProperties (java.util.Properties props) Set the JDBC connection properties.
protected void	setTablePrefix (java.lang.String tablePrefix) Set the table prefix to



use.
protected void setURL (java.lang.String url) Set the JDBC URL.
void shutdown () Close the database connection and cleanup.
Methods inherited from class com.protomatter.syslog.BasicLogger
configure , formatLogEntry , getName , getPolicy , getTextFormatter , mightLog , resetDateFormat , setName , setPolicy , setTextFormatter , shouldLog
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



Component Name: FileLog
 Classification: Java Public Class
 Definition: A logger that writes to a file.
 Uses/Interactions: Extends BasicLogger
 Implements Syslogger (indirect)
 Implements XMLConfigurable (indirect)

Interfaces/Exports:

Constructor Summary	
FileLog ()	You will need to call the configure() method if you use this constructor.
FileLog (java.io.File f)	Create a new file log attached to the given file.
FileLog (java.io.File f, boolean append, boolean autoFlush)	Create a new file log attached to the given file.
Method Summary	
void configure (org.jdom.Element e)	Configure this logger given the XML element.
boolean getAppend ()	Get the file we're writing to.
boolean GetAutoFlush ()	Determine if we should we auto-flush the buffer all the time.
org.jdom.Element GetConfiguration (org.jdom.Element element)	Get the current configuration of this logger.
java.io.File getFile ()	Get the file we're writing to.
void log (SyslogMessage message)	Log a message.
void setAppend (boolean append)	Set the file we're writing to.
void SetAutoFlush (boolean flush)	Should we auto-flush the buffer all the time?
void setFile (java.io.File f)	Set the file we're writing to.
void shutdown ()	Closes down the file and prepares for shutdown.
Methods inherited from class com.protomatter.syslog.BasicLogger	
configure , formatLogEntry , getName , getPolicy , getTextFormatter , mightLog , resetDateFormat , setName , setPolicy , setTextFormatter , shouldLog	
Methods inherited from class java.lang.Object	
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	



Component Name: HTMLSyslogTextFormatter
Classification: Java Public Class
Definition: A log entry formatter that produces HTML.
Uses/Interactions: Extends SimpleSyslogTextFormatter
 Implements SyslogTextFormatter (indirect)
 Implements XMLConfigurable (indirect)

Interfaces/Exports:

Constructor Summary	
HTMLSyslogTextFormatter	() Private constructor so nobody goes around creating these.
Method Summary	
void	configure (org.jdom.Element e) Configure this text formatter given the XML element.
void	formatLogEntry (java.lang.StringBuffer b, SyslogMessage message) Format a log entry.
org.jdom.Element	getConfiguration (org.jdom.Element element) Get this object's configuration represented as an XML Element.
java.lang.String	getLogFooter () Get the log footer.
java.lang.String	getLogHeader () Get the log header.
protected char[]	getStringForLevel (int level)
java.lang.String	getStyleSheet () Get the style sheet being used.
void	setStyleSheet (java.lang.String style sheet) Set the style sheet to use.
Methods inherited from class com.protomatter.syslog.SimpleSyslogTextFormatter	
formatDate , formatMessageDetail , getDateFormat , getDateFormatCacheTime , getDateFormatTimezone , getHostname , getNextException , getShowChannel , getShowHostName , getShowThreadName , resetDateFormat , setDateFormat , setDateFormatCacheTime , setDateFormatTimezone , setShowChannel , setShowHostName , setShowThreadName , trimFromLastPeriod , trimFromLastPeriod	
Methods inherited from class java.lang.Object	
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	



Component Name: JMSLog
Classification: Java Public Class
Definition: A logger that writes messages to JMS. The JMS session used has no transaction attribute itself, so it will obey any JTS transaction context that is currently active.
Uses/Interactions: Extends BasicLogger
 Implements JMSConstants
 Implements Syslogger (indirect)
 Implements XMLConfigurable (indirect)

Interfaces/Exports:

Fields inherited from class com.protomatter.syslog.BasicLogger
formatter , policy
Fields inherited from interface com.protomatter.syslog.JMSConstants
JMS_PROP_CHANNEL , JMS_PROP_HOST , JMS_PROP_LEVEL , JMS_PROP_LOGGER , JMS_PROP_MESSAGE , JMS_PROP_MSG_TYPE , JMS_PROP_MSG_TYPE_VALUE , JMS_PROP_THREAD , JMS_PROP_TIME
Constructor Summary
JMSLog () You will need to call the configure() method if you use this constructor.
Method Summary
void configure (org.jdom.Element e) Configure this logger given the XML element.
org.jdom.Element getConfiguration (org.jdom.Element element) Get the current configuration of this logger.
void log (SyslogMessage sm) Write a log message.
void shutdown () Prepare for shutdown.
Methods inherited from class com.protomatter.syslog.BasicLogger
configure , formatLogEntry , getName , getPolicy , getTextFormatter , mightLog , resetDateFormat , setName , setPolicy , setTextFormatter , shouldLog
Methods inherited from class java.lang.Object
clone , equals , finalize , getClass , hashCode , notify , notifyAll , toString , wait , wait , wait



Component Name: LengthRolloverLog
Classification: Java Public Class
Definition: A logger that writes to a file, and will roll its log files after a certain number of bytes have been written to them.
Uses/Interactions: Extends BasicLogger
 Implements Syslogger (indirect)
 Implements XMLConfigurable (indirect)

Interfaces/Exports:

Fields inherited from class com.protomatter.syslog.BasicLogger
formatter , policy
Constructor Summary
LengthRolloverLog () You will need to call the configure() method to configure this logger if you use this constructor.
LengthRolloverLog (java.lang.String basename, java.lang.String extension, int roll, boolean append, boolean autoFlush) Write log information to the given log, roll when specified.
Method Summary
void configure (org.jdom.Element e) Configure this logger given the XML element.
org.jdom.Element getConfiguration (org.jdom.Element element) Get the current configuration of this logger.
void log (SyslogMessage message) Log an entry to the log.
void rollover () Roll the logs now.
void shutdown () Shutdown this logger.
Methods inherited from class com.protomatter.syslog.BasicLogger
configure , formatLogEntry , getName , getPolicy , getTextFormatter , mightLog , resetDateFormat , setName , setPolicy , setTextFormatter , shouldLog
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



Component Name: MailLog
 Classification: Java Public Class
 Definition: A logger that sends email.
 Uses/Interactions: Extends BasicLogger
 Implements Syslogger (indirect)
 Implements XMLConfigurable (indirect)

Interfaces/Exports:

Fields inherited from class com.protomatter.syslog.BasicLogger	
formatter , policy	
Constructor Summary	
MailLog ()	You will need to call the configure() method if you use this constructor.
MailLog (java.lang.String workQueueName, java.lang.String smtpServer)	Create a new mail log that communicates with the given SMTP on port 25.
MailLog (java.lang.String workQueueName, java.lang.String smtpServer, int port)	Create a new mail log that communicates with the given SMTP on the given port.
Method Summary	
void configure (org.jdom.Element e)	Configure this logger given the XML element.
org.jdom.Element getConfiguration (org.jdom.Element element)	Get the current configuration of this logger.
SyslogMailSubjectFormatter getSubjectFormatter ()	Get the message subject formatter.
java.lang.String getWorkQueue ()	Get the name of the work queue that this logger will actually do work in.
void log (SyslogMessage message)	Log a message.
void setSubjectFormatter (SyslogMailSubjectFormatter subjectFormat)	Set the message subject formatter.
void setWorkQueue (java.lang.String workQueue)	Set the name of the work queue that this logger will actually do work in.
void shutdown ()	Closes down the file and prepares for shutdown.
Methods inherited from class com.protomatter.syslog.BasicLogger	
configure , formatLogEntry , getName , getPolicy , getTextFormatter , mightLog ,	



[resetDateFormat](#), [setName](#), [setPolicy](#), [setTextFormatter](#), [shouldLog](#)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait,
wait



Component Name: OpenFileLog
Classification: Java Public Class
Definition: A logger that opens the file for each log entry and closes it after it is done writing.
Uses/Interactions: Extends BasicLogger
 Implements Syslogger (indirect)
 Implements XMLConfigurable (indirect)

Interfaces/Exports:

Fields inherited from class com.protomatter.syslog.BasicLogger
formatter , policy
Constructor Summary
OpenFileLog() You will need to call the configure() method if you use this constructor.
OpenFileLog(java.io.File f) Create an OpenFileLog attached to the given file.
Method Summary
void configure (org.jdom.Element e) Configure this logger given the XML element.
org.jdom.Element getConfiguration (org.jdom.Element element) Get the current configuration of this logger.
java.io.File getFile () Get the file we're writing to.
void log (SyslogMessage message) Write a log message.
void setFile (java.io.File f) Set the file we're writing to.
void shutdown () Cleanup our file and prepare for shutdown.
Methods inherited from class com.protomatter.syslog.BasicLogger
configure , formatLogEntry , getName , getPolicy , getTextFormatter , mightLog , resetDateFormat , setName , setPolicy , setTextFormatter , shouldLog
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



Component Name: PerClassPolicy
 Classification: Java Public Class
 Definition: A policy that can make decisions on a per-class basis.
 Uses/Interactions:
 Interfaces/Exports:

Inner Class Summary	
class	PerClassPolicy.PolicyGroup A policy within a policy -- this is exactly like the SimpleLogPolicy except that it also checks to see if the class issuing the log message is in some set.
Constructor Summary	
	PerClassPolicy () Default constructor.
Method Summary	
void	addPolicyGroup (PerClassPolicy.PolicyGroup group) Add a policy group to our list.
void	configure (org.jdom.Element e) Configure this policy given the XML element.
org.jdom.Element	getConfiguration (org.jdom.Element element) Get this object's configuration represented as an XML Element.
java.util.Iterator	getPolicyGroups () Get the list of policy groups.
void	removePolicyGroup (PerClassPolicy.PolicyGroup group) Remove a policy group from our list.
boolean	shouldLog (SyslogMessage message) Decide if the message should be logged.
Methods inherited from class com.protomatter.syslog.SimpleLogPolicy	
	addChannel , getChannels , getLogMask , inMask , removeAllChannels , removeChannel , setChannels , setChannels , setLogMask , setLogMask
Methods inherited from class java.lang.Object	
	clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



Component Name: `PrintWriterLog`
Classification: Java Public Class
Definition: An implementation of an object that will log things using the Syslog facility.
Uses/Interactions: Extends `BasicLogger`
 Implements `Syslogger` (indirect)
 Implements `XMLConfigurable` (indirect)

Interfaces/Exports:

Fields inherited from class <code>com.protomatter.syslog.BasicLogger</code>
formatter , policy
Constructor Summary
PrintWriterLog () Construct a new <code>PrintWriterLog</code> -- you must call <code>configure()</code> after using this constructor.
PrintWriterLog (<code>java.io.PrintWriter</code> writer) Construct a new <code>PrintWriterLog</code> attached to the given <code>PrintWriter</code> .
PrintWriterLog (<code>java.lang.String</code> streamName) Construct a new <code>PrintWriterLog</code> attached to the given stream.
Method Summary
<code>void</code> configure (<code>org.jdom.Element</code> e) Configure this logger given the XML element.
<code>org.jdom.Element</code> getConfiguration (<code>org.jdom.Element</code> element) Get the current configuration of this logger.
<code>void</code> log (<code>SyslogMessage</code> message) Write a log message.
<code>void</code> setWriter (<code>java.io.PrintWriter</code> writer) Set the writer that we're writing to.
<code>void</code> shutdown () Clean up and prepare for shutdown.
Methods inherited from class <code>com.protomatter.syslog.BasicLogger</code>
configure , formatLogEntry , getName , getPolicy , getTextFormatter , mightLog , resetDateFormat , setName , setPolicy , setTextFormatter , shouldLog
Methods inherited from class <code>java.lang.Object</code>
<code>clone</code> , <code>equals</code> , <code>finalize</code> , <code>getClass</code> , <code>hashCode</code> , <code>notify</code> , <code>notifyAll</code> , <code>toString</code> , <code>wait</code> , <code>wait</code> , <code>wait</code>



Component Name: RemoteLog
Classification: Java Public Class
Definition: A logger that sends messages to remote receivers bound in JNDI. Objects bound directly under the 'com.protomatter.syslog.remote' location in JNDI will receive the log message if they implement the RemoteLogReceiver interface.
Uses/Interactions: Extends BasicLogger
 Implements Syslogger (indirect)
 Implements XMLConfigurable (indirect)

Interfaces/Exports:

Fields inherited from class com.protomatter.syslog.BasicLogger
formatter , policy
Constructor Summary
RemoteLog () You will need to call the configure() method if you use this constructor.
Method Summary
void configure (org.jdom.Element e) Configure this logger given the XML element.
org.jdom.Element getConfiguration (org.jdom.Element element) Get the current configuration of this logger.
void log (SyslogMessage sm) Log the given message to all bound listeners.
void shutdown () Prepare for shutdown.
Methods inherited from class com.protomatter.syslog.BasicLogger
configure , formatLogEntry , getName , getPolicy , getTextFormatter , mightLog , resetDateFormat , setName , setPolicy , setTextFormatter , shouldLog
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



Component Name: RemoteLogReceiverImpl
Classification: Java Public Class
Definition: A simple implementation of the RemoteLogReceiver interface. The SyslogServer class uses this object if it is listening to remote messages (not messages from a JMS topic, though).

Uses/Interactions:

Interfaces/Exports:

Constructor Summary
RemoteLogReceiverImpl() Default constructor.
Method Summary
<pre>void log(java.lang.String ipAddress, java.lang.String loggerClass, java.lang.String channel, java.lang.String message, java.lang.Object detail, int level, java.lang.String threadName, long messageSendTime) Remote log receiver callback.</pre>
Methods inherited from class java.lang.Object
<pre>clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait</pre>



Component Name: SimpleLogPolicy
Classification: Java Public Class
Definition: The default LogPolicy that knows about log levels and channels.
Uses/Interactions:
Interfaces/Exports:

Constructor Summary	
SimpleLogPolicy()	All channels are listened to by default, and the default log mask is inherited from Syslog itself.
Method Summary	
void addChannel (java.lang.String channel)	Add the given channel to the list of channels we are listening to.
void configure (org.jdom.Element e)	Configure this policy given the XML element.
java.util.Iterator getChannels ()	Get the list of channels this policy listens to.
org.jdom.Element getConfiguration (org.jdom.Element element)	Get this object's configuration represented as an XML Element.
int getLogMask ()	Get the mask for logging of messages.
protected boolean inMask (int level, int mask)	Check if the given level is covered by the given mask.
void removeAllChannels ()	Remove all channels from the list of channels we are listening to.
void removeChannel (java.lang.String channel)	Remove the given channel from the list of channels we are listening to.
void setChannels (java.util.List channels)	Set the list of channels to use.
void setChannels (java.util.Set channelSet)	Set the list of channels to use.
void setLogMask (int mask)	Set the mask for logging of messages.
void setLogMask (java.lang.String minLevel)	Set the mask to at or above the level specified.
boolean shouldLog (SyslogMessage message)	Determine if a log message should be logged given the information.
Methods inherited from class java.lang.Object	
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	



Component Name: SimpleSyslogMailSubjectFormatter
Classification: Java Public Class
Definition: A simple mail subject formatter. This class is used by default by the MailLog logger.
Uses/Interactions: Implements SyslogMailSubjectFormatter
 Implements XMLConfigurable (indirect)
Interfaces/Exports:

Constructor Summary	
SimpleSyslogMailSubjectFormatter	() Default constructor.
Method Summary	
void	configure (org.jdom.Element e) Configure this text formatter given the XML element.
java.lang.String	formatMessageSubject (SyslogMessage message) Format the given log entry.
org.jdom.Element	getConfiguration (org.jdom.Element element) Get this object's configuration represented as an XML Element.
protected java.lang.String	getHostname (java.net.InetAddress host)
boolean	getShowChannel () Get whether we should show the channel name in the output.
boolean	getShowHostName () Get whether we should show the host name in the output.
boolean	getShowThreadName () Get whether we should show the thread name in the output.
protected java.lang.String	getStringForLevel (int level)
void	setShowChannel (boolean showChannel) Set whether we should show the channel name in the output.
void	setShowHostName (boolean showHostName) Set whether we should show the host name in the output.
void	setShowThreadName (boolean showThreadName) Set whether we should show the thread name in the output.
protected java.lang.String	trimFromLastPeriod (java.lang.String s) Given something like "foo.bar.Baz" this will return "Baz".
Methods inherited from class java.lang.Object	
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	



Component Name: SimpleSyslogTextFormatter
Classification: Java Public Class
Definition: A simple log entry formatter. This class is used by several of the included Syslogger implementations to format their log entries.
Uses/Interactions: Implements SyslogTextFormatter
 Implements XMLConfigurable (indirect)
 Extended by HTMLSyslogTextFormatter
 Extended by SyslogHTMLMailFormatter
 Extended by WISyslogTextFormatter

Interfaces/Exports:

Constructor Summary	
SimpleSyslogTextFormatter()	Default constructor.
Method Summary	
void configure (org.jdom.Element e)	Configure this text formatter given the XML element.
protected char[] formatDate (long theDate)	Format the given date with the dateformat that's been set.
void formatLogEntry (java.lang.StringBuffer b, SyslogMessage message)	Format the given log entry.
void formatMessageDetail (java.lang.StringBuffer b, SyslogMessage message)	Format the given log message's detail.
org.jdom.Element getConfiguration (org.jdom.Element element)	Get this object's configuration represented as an XML Element.
java.lang.String getDateFormat ()	Get the format for logging dates.
int getDateFormatCacheTime ()	Get the number of milliseconds a date format should be cached.
java.util.TimeZone getDateFormatTimezone ()	Get the timezone of the date format.
protected java.lang.String getHostname (java.net.InetAddress host)	
java.lang.String getLogFooter ()	Get the log footer.
java.lang.String getLogHeader ()	Get the log header.
protected java.lang.Object[] getNextException (java.lang.Throwable t)	Get the "next" exception in this series.



<code>boolean getShowChannel()</code> Get whether we should show the channel name in the output.
<code>boolean getShowHostName()</code> Get whether we should show the host name in the output.
<code>boolean getShowThreadName()</code> Get whether we should show the thread name in the output.
<code>protected char[] getStringForLevel(int level)</code>
<code>void resetDateFormat()</code> Reset the <code>formatDate(...)</code> method so that it's guaranteed to not return a cached date string the next time it's called.
<code>void setDateFormat(java.lang.String format)</code> Set the format for logging dates.
<code>void setDateFormatCacheTime(int cacheTime)</code> Set the number of milliseconds a date format should be cached.
<code>void setDateFormatTimezone(java.util.TimeZone zone)</code> Set the time zone of the date format.
<code>void setShowChannel(boolean showChannel)</code> Set whether we should show the channel name in the output.
<code>void setShowHostName(boolean showHostName)</code> Set whether we should show the host name in the output.
<code>void setShowThreadName(boolean showThreadName)</code> Set whether we should show the thread name in the output.
<code>protected java.lang.String trimFromLastPeriod(java.lang.String s)</code>
<code>protected void trimFromLastPeriod(java.lang.StringBuffer b, java.lang.String s, int width)</code> Given something like "foo.bar.Baz" this will return "Baz".
Methods inherited from class java.lang.Object
<code>clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait</code>



Component Name: Syslog
Classification: Java Public Class
Definition: This class implements a system-wide logging utility. It allows a program to log messages and objects and different severity levels in a standardized way.
Uses/Interactions:
Interfaces/Exports:

Field Summary	
static java.lang.String	ALL_CHANNEL The symbolic name for all channels.
static int	currentLogMask The current system-wide log mask.
static int	DEBUG A log generated during debugging of the software.
static java.lang.String	DEFAULT_CHANNEL The name of the default log channel.
static int	ERROR One of the software components caused an error or exception.
static int	FATAL One of the software components is no longer functional.
static int	INFO An informational message that might come in handy later.
static int	INHERIT_MASK Loggers can inherit Syslog's log mask by setting their log mask to this value.
static int	WARNING A warning message that the system administrator might want to know about.
Method Summary	
static void	addLogger (Syslogger log) Registers a new Syslogger object with Syslog.
static void	addWork (java.lang.String queueName, java.lang.Runnable r) Add work to the given background queue.
static int	atOrAbove (int level)
static boolean	canDebug () Determine if the current syslog mask would allow a message at the DEBUG level to be logged.
static boolean	canDebug (java.lang.Object logger) Deprecated.
static boolean	canError () Determine if the current syslog mask would allow a message at the ERROR level to be logged.
static boolean	canError (java.lang.Object logger) Deprecated.



<code>static boolean canFatal()</code> Determine if the current syslog mask would allow a message at the FATAL level to be logged.
<code>static boolean canFatal(java.lang.Object logger)</code> Deprecated.
<code>static boolean canInfo()</code> Determine if the current syslog mask would allow a message at the INFO level to be logged.
<code>static boolean canInfo(java.lang.Object logger)</code> Deprecated.
<code>static boolean canLog(int level)</code> Determine if the current default syslog mask would allow the given level of message to be logged.
<code>static boolean canLog(java.lang.Object logger, int level)</code> Deprecated.
<code>static boolean canWarning()</code> Determine if the current syslog mask would allow a message at the WARNING level to be logged.
<code>static boolean canWarning(java.lang.Object logger)</code> Deprecated.
<code>static boolean configure(org.jdom.Element syslogConfig)</code> Configure syslog from the given XML document.
<code>static boolean configure(java.io.File xmlFile)</code> Configure syslog from the given XML file.
<code>static void debug(java.net.InetAddress host, java.lang.Object logger, java.lang.Object message)</code> Logs a debug message, which will be converted through <code>toString()</code> .
<code>static void debug(java.net.InetAddress host, java.lang.Object logger, java.lang.Object message, java.lang.Object detail)</code> Logs a debug message with a detail object, both of which will be converted through <code>toString()</code> .
<code>static void debug(java.lang.Object logger, java.lang.Object message)</code> Logs a debug message, which will be converted through <code>toString()</code> .
<code>static void debug(java.lang.Object logger, java.lang.Object message, java.lang.Object detail)</code> Logs a debug message with a detail object, both of which will be converted through <code>toString()</code> .
<code>static void debugToChannel(java.net.InetAddress host, java.lang.Object logger, java.lang.Object channel, java.lang.Object message)</code> Logs a debug message to the given channel.
<code>static void debugToChannel(java.net.InetAddress host, java.lang.Object logger, java.lang.Object channel, java.lang.Object message, java.lang.Object detail)</code> Logs a debug message with a detail object to the given channel.
<code>static void debugToChannel(java.lang.Object logger, java.lang.Object channel, java.lang.Object message)</code> Logs a debug message to the given channel.



<code>java.lang.Object message)</code> Logs a debug message to the given channel.
static void debugToChannel (java.lang.Object logger, java.lang.Object channel, java.lang.Object message, java.lang.Object detail) Logs a debug message with a detail object to the given channel.
static void error (java.net.InetAddress host, java.lang.Object logger, java.lang.Object message) Logs an error message, which will be converted through toString().
static void error (java.net.InetAddress host, java.lang.Object logger, java.lang.Object message, java.lang.Object detail) Logs an error message with a detail object, both of which will be converted through toString().
static void error (java.lang.Object logger, java.lang.Object message) Logs an error message, which will be converted through toString().
static void error (java.lang.Object logger, java.lang.Object message, java.lang.Object detail) Logs an error message with a detail object, both of which will be converted through toString().
static void errorToChannel (java.net.InetAddress host, java.lang.Object logger, java.lang.Object channel, java.lang.Object message) Logs an error message to the given channel.
static void errorToChannel (java.net.InetAddress host, java.lang.Object logger, java.lang.Object channel, java.lang.Object message, java.lang.Object detail) Logs an error message with a detail object to the given channel.
static void errorToChannel (java.lang.Object logger, java.lang.Object channel, java.lang.Object message) Logs an error message to the given channel.
static void errorToChannel (java.lang.Object logger, java.lang.Object channel, java.lang.Object message, java.lang.Object detail) Logs an error message with a detail object to the given channel.
static void fatal (java.net.InetAddress host, java.lang.Object logger, java.lang.Object message) Logs a fatal message, which will be converted through toString().
static void fatal (java.net.InetAddress host, java.lang.Object logger, java.lang.Object message, java.lang.Object detail) Logs a fatal message with a detail object, both of which will be converted through toString().
static void fatal (java.lang.Object logger, java.lang.Object message) Logs a fatal message, which will be converted through toString().
static void fatal (java.lang.Object logger, java.lang.Object message, java.lang.Object detail) Logs a fatal message with a detail object, both of which will be converted through toString().



<pre>static void fatalToChannel(java.net.InetAddress host, java.lang.Object logger, java.lang.Object channel, java.lang.Object message) Logs a fatal message to the given channel.</pre>
<pre>static void fatalToChannel(java.net.InetAddress host, java.lang.Object logger, java.lang.Object channel, java.lang.Object message, java.lang.Object detail) Logs a fatal message with a detail object to the given channel.</pre>
<pre>static void fatalToChannel(java.lang.Object logger, java.lang.Object channel, java.lang.Object message) Logs a fatal message to the given channel.</pre>
<pre>static void fatalToChannel(java.lang.Object logger, java.lang.Object channel, java.lang.Object message, java.lang.Object detail) Logs a fatal message with a detail object to the given channel.</pre>
<pre>static org.jdom.Docume getConfiguration() Get an XML representation of the current configuration state of Syslog nt including all loggers, etc.</pre>
<pre>static Syslog getInstance() Returns the global Syslog instance.</pre>
<pre>static java.net.InetAd getLocalHostName() Get the local hostname. dress</pre>
<pre>static Syslogger getLogger(java.lang.String name) Get a logger by name.</pre>
<pre>static java.util.Itera getLoggers() Returns an Enumeration of all Syslogger objects registered with Syslog. tor</pre>
<pre>static int getLogMask() Get the default mask for logging of messages.</pre>
<pre>static java.util.Map getWorkQueueMap() Get a map of the background work queues.</pre>
<pre>static void info(java.net.InetAddress host, java.lang.Object logger, java.lang.Object message) Logs an info message, which will be converted through toString().</pre>
<pre>static void info(java.net.InetAddress host, java.lang.Object logger, java.lang.Object message, java.lang.Object detail) Logs an info message with a detail object, both of which will be converted through toString().</pre>
<pre>static void info(java.lang.Object logger, java.lang.Object message) Logs an info message, which will be converted through toString().</pre>
<pre>static void info(java.lang.Object logger, java.lang.Object message, java.lang.Object detail) Logs an info message with a detail object, both of which will be converted through toString().</pre>



<pre>static void infoToChannel(java.net.InetAddress host, java.lang.Object logger, java.lang.Object channel, java.lang.Object message) Logs an info message to the given channel.</pre>
<pre>static void infoToChannel(java.net.InetAddress host, java.lang.Object logger, java.lang.Object channel, java.lang.Object message, java.lang.Object detail) Logs an info message with a detail object to the given channel.</pre>
<pre>static void infoToChannel(java.lang.Object logger, java.lang.Object channel, java.lang.Object message) Logs an info message to the given channel.</pre>
<pre>static void infoToChannel(java.lang.Object logger, java.lang.Object channel, java.lang.Object message, java.lang.Object detail) Logs an info message with a detail object to the given channel.</pre>
<pre>static void log(java.net.InetAddress host, java.lang.Object logger, java.lang.Object msg, java.lang.Object detail, int level) Log a message.</pre>
<pre>static void log(java.net.InetAddress host, java.lang.Object logger, java.lang.Object channel, java.lang.Object msg, java.lang.Object detail, int level) Log a message.</pre>
<pre>static void log(java.net.InetAddress host, java.lang.Object logger, java.lang.Object channel, java.lang.Object msg, java.lang.Object detail, int level, java.lang.Thread thread, java.lang.String threadName, long messageSendTime) Log a message.</pre>
<pre>static void log(java.net.InetAddress host, java.lang.Object logger, java.lang.Throwable e) Logs an exception for the given object.</pre>
<pre>static void log(java.net.InetAddress host, java.lang.Object logger, java.lang.Throwable e, int level) Logs an exception for the given object at a given level.</pre>
<pre>static void log(java.lang.Object logger, java.lang.Object msg, java.lang.Object detail, int level) Log a message.</pre>
<pre>static void log(java.lang.Object logger, java.lang.Object channel, java.lang.Object msg, java.lang.Object detail, int level) Log a message.</pre>
<pre>static void log(java.lang.Object logger, java.lang.Throwable e) Logs an exception for the given object.</pre>
<pre>static void log(java.lang.Object logger, java.lang.Throwable e, int level) Logs an exception for the given object at a given level.</pre>
<pre>static void main(java.lang.String[] args) Write an example XML configuration file to standard out.</pre>
<pre>static boolean mightDebug(java.lang.Object logger) Determine if it's likely that someone will listen to a debug message from the given logger.</pre>



<code>static boolean mightDebug(java.lang.Object logger, java.lang.Object channel)</code> Determine if it's likely that someone will listen to a debug message from the given logger on the given channel(s).
<code>static boolean mightError(java.lang.Object logger)</code> Determine if it's likely that someone will listen to an error message from the given logger.
<code>static boolean mightError(java.lang.Object logger, java.lang.Object channel)</code> Determine if it's likely that someone will listen to an error message from the given logger on the given channel(s).
<code>static boolean mightFatal(java.lang.Object logger)</code> Determine if it's likely that someone will listen to a fatal message from the given logger.
<code>static boolean mightFatal(java.lang.Object logger, java.lang.Object channel)</code> Determine if it's likely that someone will listen to a fatal message from the given logger on the given channel(s).
<code>static boolean mightInfo(java.lang.Object logger)</code> Determine if it's likely that someone will listen to an info message from the given logger.
<code>static boolean mightInfo(java.lang.Object logger, java.lang.Object channel)</code> Determine if it's likely that someone will listen to an info message from the given logger on the given channel(s).
<code>static boolean mightLog(java.lang.Object logger, int level)</code> Determine if it's likely that someone will listen to a message from the given logger at the given level.
<code>static boolean mightLog(java.lang.Object logger, int level, java.lang.Object channel)</code> Determine if it's likely that someone will listen to a message from the given logger at the given level on the given channel(s).
<code>static boolean mightWarning(java.lang.Object logger)</code> Determine if it's likely that someone will listen to a warning message from the given logger.
<code>static boolean mightWarning(java.lang.Object logger, java.lang.Object channel)</code> Determine if it's likely that someone will listen to a warning message from the given logger on the given channel(s).
<code>static void removeAllLoggers()</code> Deregisters all Syslogger objects.
<code>static boolean removeLogger(Syslogger log)</code> Deregisters a Syslogger object from Syslog.
<code>static void setLocalHostName()</code> Set the local hostname automatically.
<code>static void setLocalHostName(java.net.InetAddress host)</code> Set the local hostname.
<code>static void setLogMask(int mask)</code> Set the default mask for logging of messages.



static void setLogMask (java.lang.String minLevel) Set the mask to at or above the level specified.
static void shutdown () Remove all the loggers and shut them down.
static void warning (java.net.InetAddress host, java.lang.Object logger, java.lang.Object message) Logs a warning message, which will be converted through toString().
static void warning (java.net.InetAddress host, java.lang.Object logger, java.lang.Object message, java.lang.Object detail) Logs a warning message with a detail object, both of which will be converted through toString().
static void warning (java.lang.Object logger, java.lang.Object message) Logs a warning message, which will be converted through toString().
static void warning (java.lang.Object logger, java.lang.Object message, java.lang.Object detail) Logs a warning message with a detail object, both of which will be converted through toString().
static void warningToChannel (java.net.InetAddress host, java.lang.Object logger, java.lang.Object channel, java.lang.Object message) Logs a warning message to the given channel.
static void warningToChannel (java.net.InetAddress host, java.lang.Object logger, java.lang.Object channel, java.lang.Object message, java.lang.Object detail) Logs a warning message with a detail object to the given channel.
static void warningToChannel (java.lang.Object logger, java.lang.Object channel, java.lang.Object message) Logs a warning message to the given channel.
static void warningToChannel (java.lang.Object logger, java.lang.Object channel, java.lang.Object message, java.lang.Object detail) Logs a warning message with a detail object to the given channel.
static void writeConfiguration (java.io.OutputStream out) Get an XML representation of the current configuration state of Syslog including all loggers, etc and write it to the given output stream.
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



Component Name: SyslogHTMLMailFormatter
 Classification: Java Public Class
 Definition: A simple HTML log entry formatter for email.
 Uses/Interactions: Extends SimpleSyslogTextFormatter
 Implements SyslogTextFormatter (indirect)
 Implements XMLConfigurable (indirect)

Interfaces/Exports:

Constructor Summary
SyslogHTMLMailFormatter () Default constructor.
Method Summary
java.lang.String formatLogEntry (SyslogMessage message) Format the given log entry.
protected char[] getStringForLevel (int level)
Methods inherited from class com.protomatter.syslog.SimpleSyslogTextFormatter
configure , formatDate , formatLogEntry , formatMessageDetail , getConfiguration , getDateFormat , getDateFormatCacheTime , getDateFormatTimezone , getHostname , getLogFooter , getLogHeader , getNextException , getShowChannel , getShowHostName , getShowThreadName , resetDateFormat , setDateFormat , setDateFormatCacheTime , setDateFormatTimezone , setShowChannel , setShowHostName , setShowThreadName , trimFromLastPeriod , trimFromLastPeriod
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



Component Name: SyslogMessage
Classification: Java Public Class
Definition: A utility class representing all the information needed to make a syslog call.

Uses/Interactions:

Interfaces/Exports:

Field Summary	
java.lang.String	channel The channel the message is for.
java.lang.Object	detail The detailed message.
java.net.InetAddress	host The address of the host making the call.
int	level The log level.
java.lang.Object	logger The object making the syslog call.
java.lang.String	loggerClassname The classname of the logger.
java.lang.Object	msg The message.
java.lang.Thread	thread The thread that made the log request.
java.lang.String	threadName The output of toString() on the thread that made the log request.
long	time The time the call was made.
Constructor Summary	
SyslogMessage() Default constructor.	
SyslogMessage (java.net.InetAddress host, long time, java.lang.String channel, java.lang.Object logger, java.lang.String loggerClassname, java.lang.Object msg, java.lang.Object detail, int level, java.lang.Thread thread, java.lang.String threadName) A utility constructor.	
Methods inherited from class java.lang.Object	
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	



Component Name: SyslogServer
Classification: Java Public Class
Definition: A standalone log processing server that either reads messages from a JMS topic, or through RMI.

Uses/Interactions:

Interfaces/Exports:

Method Summary
<code>static void main(java.lang.String[] args)</code> Start the syslog log server.
Methods inherited from class java.lang.Object
<code>clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait</code>



Component Name: SyslogT3Startup
 Classification: Java Public Class
 Definition: Configure syslog to start when WebLogic does.
 Uses/Interactions:
 Interfaces/Exports:

Constructor Summary	
SyslogT3Startup()	Default constructor -- called by WebLogic.
Method Summary	
void	setServices (weblogic.common.T3ServicesDef services) Part of the weblogic.common.T3StartupDef interface.
java.lang.String	shutdown (java.lang.String name, java.util.Hashtable ht) Shutdown Syslog services.
java.lang.String	startup (java.lang.String name, java.util.Hashtable ht) Start Syslog services.
boolean	startup (weblogic.common.T3ServicesDef services) A shortcut to starting syslog services.
Methods inherited from class java.lang.Object	
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	



Component Name: SyslogWriter
Classification: Java Public Class
Definition: A writer that is attached to Syslog. When data is flushed, any information built up is sent off to Syslog.

Uses/Interactions:

Interfaces/Exports:

Fields inherited from class java.io.Writer
lock
Constructor Summary
SyslogWriter (java.lang.Object logger, int level) Create a new SyslogWriter.
SyslogWriter (java.lang.Object logger, java.lang.Object channel, int level) Create a new SyslogWriter.
Method Summary
void close () Close the writer.
void flush () Flush unwritten data to Syslog.
void write (char[] buf, int offset, int length) Write the given data to the writer.
Methods inherited from class java.io.Writer
write, write, write, write
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



Component Name: TimeRolloverLog
Classification: Java Public Class
Definition: An implementation of an object that will log things using the Syslog facility, and roll its log files after a certain amount of time has passed.
Uses/Interactions: Extends BasicLogger
 Implements Syslogger (indirect)
 Implements XMLConfigurable (indirect)

Interfaces/Exports:

Field Summary	
static int	ROLL_DAILY Roll logs at midnight.
static int	ROLL_HOURLY Roll logs on the hour.
static int	ROLL_MINUTELY Roll logs on the minute.
static int	ROLL_MONTHLY Roll logs at midnight at the end of the month.
Fields inherited from class com.protomatter.syslog. BasicLogger	
formatter , policy	
Constructor Summary	
TimeRolloverLog ()	You must call the configure() method to configure this logger if you use this constructor.
TimeRolloverLog (java.lang.String basename, int roll, java.lang.String extension)	Write log information to the given log, roll when specified.
TimeRolloverLog (java.lang.String basename, java.lang.String extension, int roll, java.lang.String nameformat, boolean append, boolean autoFlush)	Write log information to the given log, roll when specified.
Method Summary	
void	configure (org.jdom.Element e) Configure this logger given the XML element.
boolean	getAppend () Determine if we will append to files that already exist.
boolean	getAutoFlush () Determine if we will automatically flush the writer.
java.lang.String	getBaseFilename () Get the base name for the log file.
org.jdom.Element	getConfiguration (org.jdom.Element element) Get the current configuration of this logger.
java.lang.String	getFileExtension () Get the file extension to use.
java.lang.String	getNameFormat () Get the dateformat part of the name.
java.util.Date	getNextRolloverTime () Calculate the next date to rollover the logs based on how often we should roll.
int	getRollType () Get the roll type.
void	log (SyslogMessage message) Log an entry to the log.
void	setAutoFlush (boolean flush) Should we will automatically flush the writer?
void	setNameFormat (java.lang.String fmt) Set the dateformat part of



the name.
void shutdown() Cleanup and prepare for shutdown.
Methods inherited from class com.protomatter.syslog.BasicLogger
configure , formatLogEntry , getName , getPolicy , getTextFormatter , mightLog , resetDateFormat , setName , setPolicy , setTextFormatter , shouldLog
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



Component Name: WISyslogTextFormatter
 Classification: Java Public Class
 Definition: A log formatter that mimics the log output from WebLogic.
 Uses/Interactions: Extends SimpleSyslogTextFormatter
 Implements SyslogTextFormatter (indirect)
 Implements XMLConfigurable (indirect)

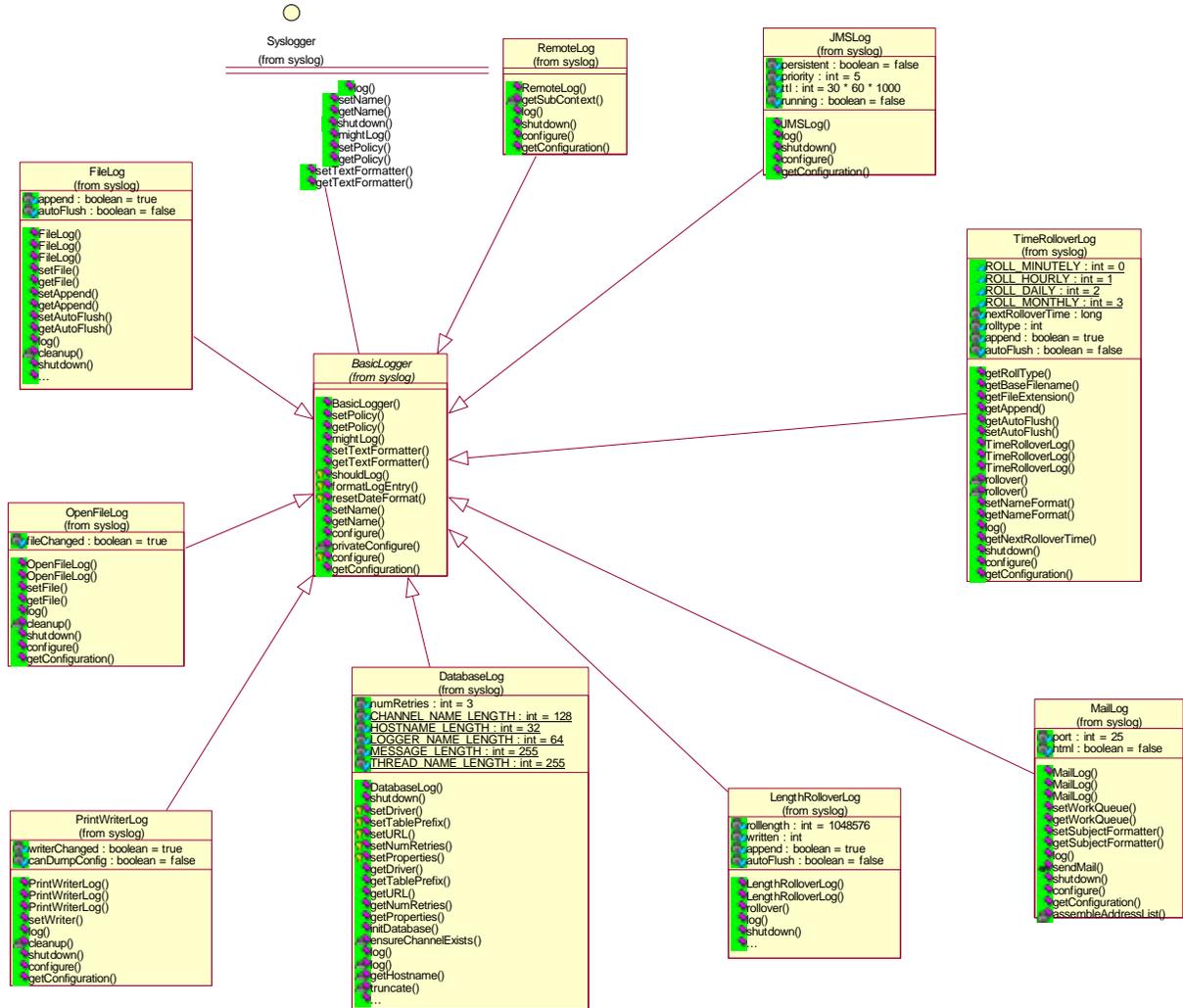
Interfaces/Exports:

Constructor Summary
WISyslogTextFormatter()
Method Summary
java.lang.String formatLogEntry(SyslogMessage message)
protected char[] getStringForLevel(int level)
Methods inherited from class com.protomatter.syslog.SimpleSyslogTextFormatter
configure , formatDate , formatLogEntry , formatMessageDetail , getConfiguration , getDateFormat , getDateFormatCacheTime , getDateFormatTimezone , getHostname , getLogFooter , getLogHeader , getNextException , getShowChannel , getShowHostName , getShowThreadName , resetDateFormat , setDateFormat , setDateFormatCacheTime , setDateFormatTimezone , setShowChannel , setShowHostName , setShowThreadName , trimFromLastPeriod , trimFromLastPeriod
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



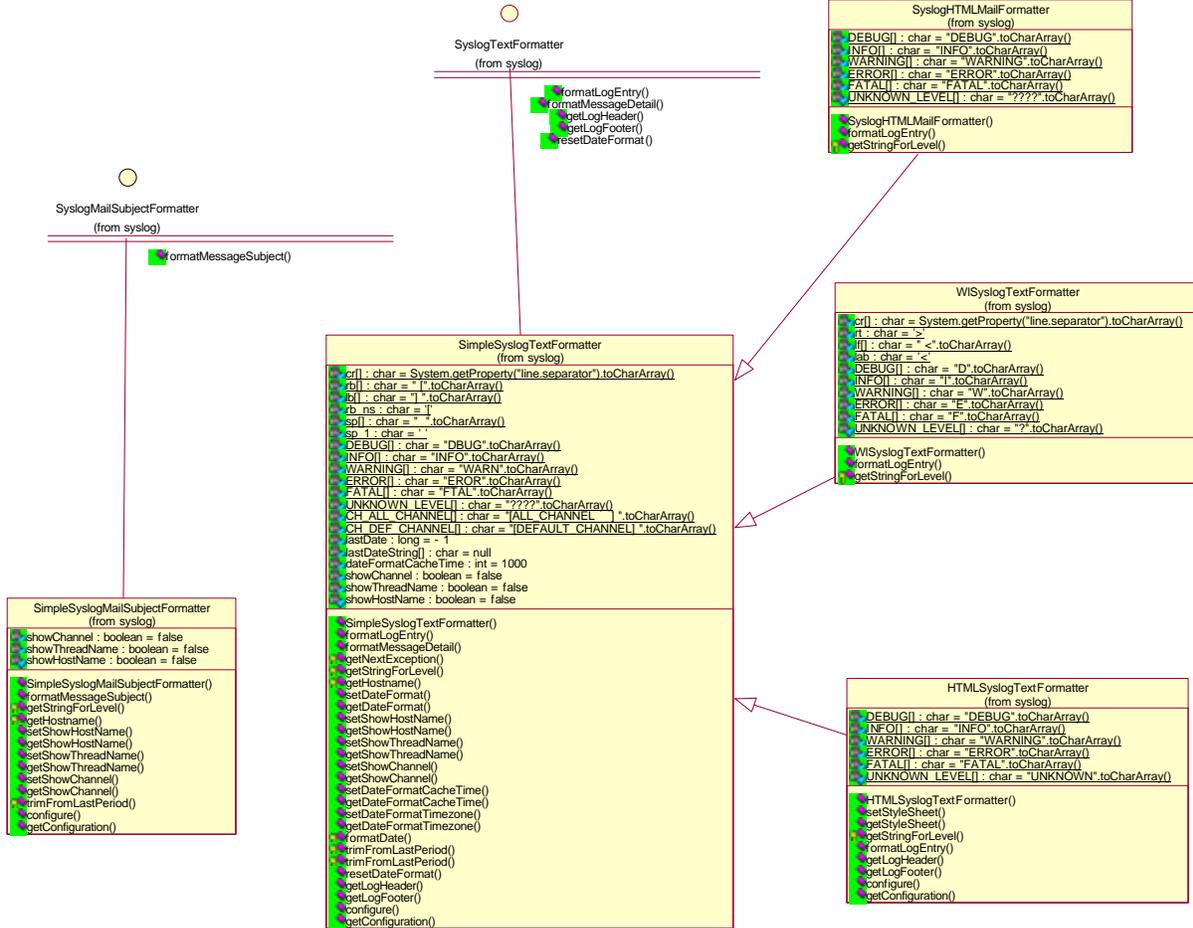
3.4.6 Class Diagrams

3.4.6.1 Logger Classes





3.4.6.2 Formatter Classes





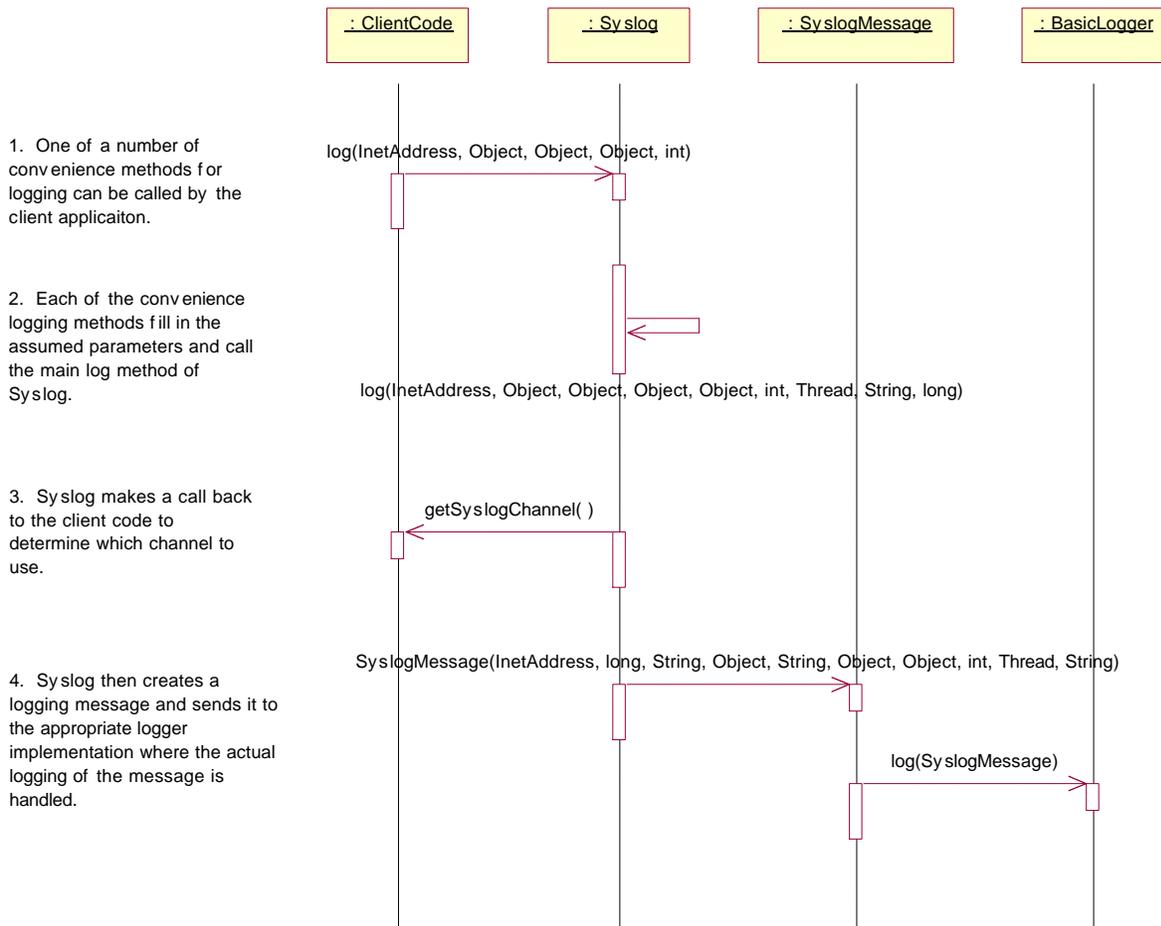
3.4.6.3 Policy Classes





3.4.7 Interaction Diagrams

This sequence diagram illustrates the main object interactions involved in a basic log() call from client code. Most other calls such as debug(), info(), warning() error(), fatal, and xxxToChannel() basically serve as utility wrappers for the basic log() call. Understanding any one of the wrapper calls, and the flow through to the formatting and logging, should serve as solid basis for all other logging-type calls in this package.





3.4.8 References

- Protomatter web-site JavaDoc on Protomatter API
<http://protomatter.sourceforge.net/latest/javadoc/com/protomatter/syslog/package-summary.html>
- Syslog white paper
<http://protomatter.sourceforge.net/latest/javadoc/com/protomatter/syslog/syslog-whitepaper.html>



3.5 Persistence

3.5.1 Introduction

The Java persistence framework provides services that interact with application databases to create, retrieve, update, and delete business objects. The framework will also provide support for ad-hoc database interaction (interacting with databases outside of the business object framework).

The ITA persistence feature set is implemented as an SFA-customized version of the Accenture ReTA (Reusable eCommerce Technical Architecture) persistence framework. The ReTA code base has been significantly modified to meet SFA application development requirements, and to work optimally in the SFA technical environment. The basis for this framework is the ReTA code, however, the ITA Persistence feature is now the standard. Specifically, the framework has been modified to:

- Support applications running on the IBM WebSphere Application Server
- Implement IBM WAS best practices for database access
- Implement IBM WAS best practices for object management
- Integrate the ITA Exception Handling framework
- Integrate the ITA Logging framework
- Support ad-hoc database interaction

3.5.2 System Overview

The persistence framework provides a transparent and flexible mapping of the business objects to relational database tables. It is transparent in that once the business objects and their mappings are defined, application developers do not need to have any knowledge of the underlying relational database tables. It is flexible in that if the underlying relational database model changes, the business object model does not have to change with it – a change in the mapping layer is all that should be required. The framework is made up of several components working together:

- Domain Component
- Unit of Work Component
- Persistable Object Manager Component
- Result Set Component
- Business Mapper Component
- Business Object Component



A diagram of the components working together (retrieving a Customer Object from the database) is presented below.

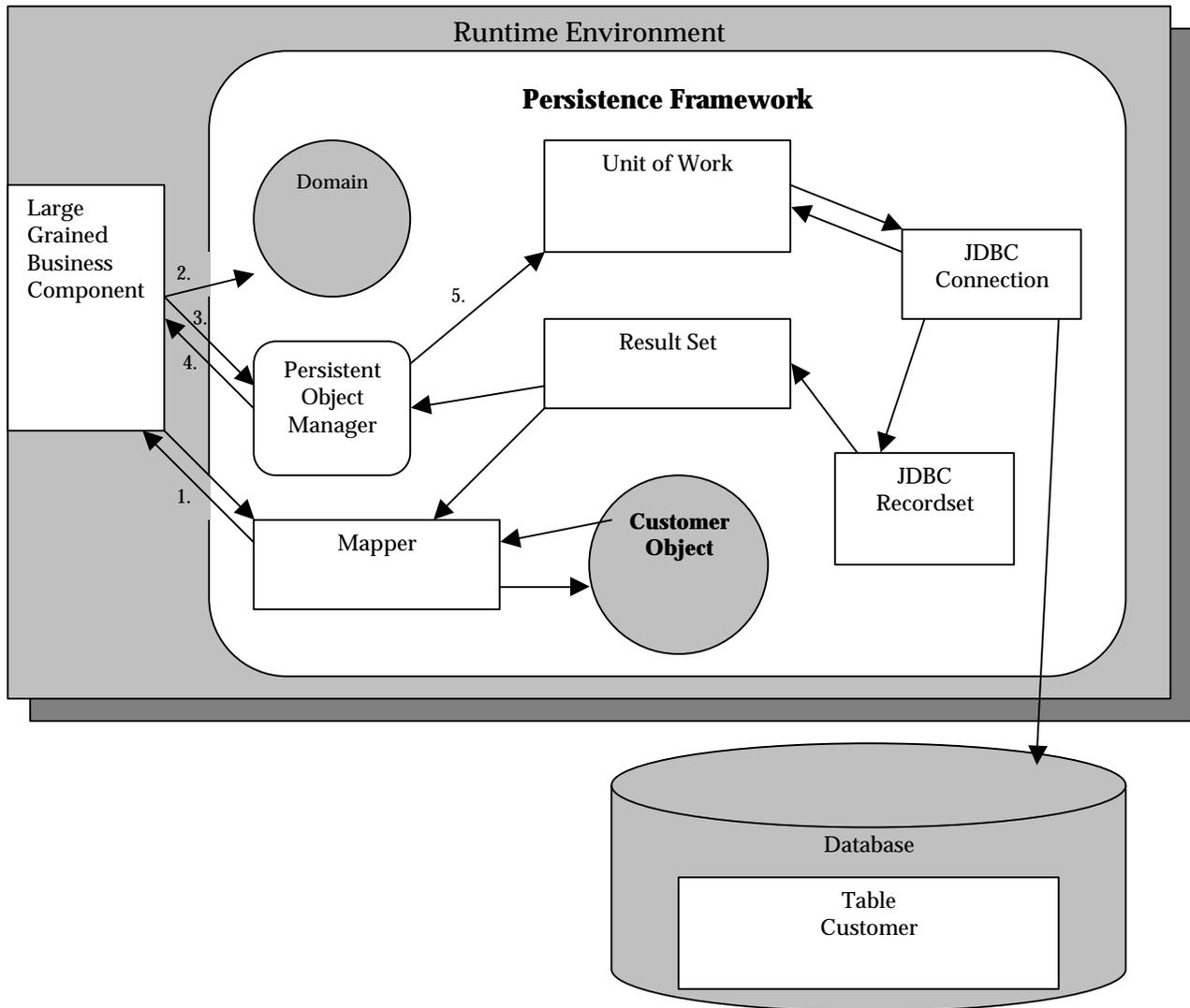


Figure 7: Using the ITA Persistence framework



Explanation of Steps

1. Create Mapper and Business Object, populate values to update
2. Create Domain with connection information
3. Create Persistable Object Manager passing the domain
4. Request object from the POM passing the Mapper information
5. POM executes the query, maps the data to the object, closes the unit of work, and returns the object

3.5.3 Design Considerations

3.5.3.1 Assumptions and Dependencies

It is assumed that this framework will function in a J2EE application server environment. As the current production server for SFA is IBM WebSphere 3.5, the framework will be compiled using its required JDK version 1.2.2. It should also work with the current JavaServer Pages (1.1), Java Servlet (2.2), Java Messaging Service (1.0.1), and Java Database Connectivity (2.0) specifications for this server.

3.5.3.2 Goals and Guidelines

The goal of this development was to provide a simple yet robust persistence framework that could easily be utilized by any SFA development team building applications in Java, or more specifically on the WebSphere Application Server. It should be noted however, that while the design of the persistence framework does incorporate the best practices as recommended in WebSphere documentation, there is nothing in this package that ties it to WebSphere. An important part of any project, database access is one of the most time-consuming coding tasks and one of the most resource intensive components of a deployed application. Correct database access coding is critical to the performance and maintainability of an application. However, without a framework in place, each developer on a project will code database access as he or she sees fit (and best knows how), often leading to a haphazard implementation and duplication of effort. As such, object persistence and database access should be addressed with care and forethought.

3.5.3.3 Development Methods

This framework was developed using general object-oriented software development techniques as are specified in any standard text on the Java programming language. As the framework itself is fairly straightforward in its class and relationship patterns it employs, no object oriented modeling tool or methodology was specifically used in its design. However, the resulting source code has been documented with standard class diagrams and sequence diagrams in order to illustrate its structure more readily. These diagrams should be of great help to programmers unfamiliar with this framework.



3.5.4 System Architecture

3.5.4.1 Overview

The Persistence Framework provides the following services:

- **Database Connection:** the database connection is uncoupled from the application, and optimal connection techniques (such as database pooling) are used to minimize server resources.
- **Database Mapping:** the business objects are mapped to database table(s) for data transparency and flexibility in application development
- **Object Query:** the queries on objects are triggered from business object events
- **Record Locking:** optimistic locking of retrieved data is implemented, in order to preserve integrity of data

3.5.4.2 Subsystem Architecture: Domain

The Domain is an object store that holds a database URL, a username, and a password. It represents the domain that the data resides in for the application. Storing database connection information in this manner supports ease of application configuration updates.

3.5.4.3 Subsystem Architecture: Unit of Work

The UnitOfWork models a persistence unit of work. A UnitOfWork has a Connection that represents the actual physical connection to the database. It allows a series of related actions to be logically grouped together that can be either committed or rolled back against a database connection. A UnitOfWork is also responsible for managing object identity conflicts and caching of business objects, if it is supported by the Persistable objects.

3.5.4.4 Subsystem Architecture: Persistable Object Manager

The Persistable Object Manager is used to create, read, update, and delete information from the data store. It holds (or creates and holds) a single Unit of Work that can be used throughout a Business Component method that accesses the data store.

Methods from the BusinessComponent class and the Business class that called methods within this class should create a single PersistableObjectManager object and directly call the methods here for persistent data access and storage. The PersistableObjectManager class also acts as a wrapper for the data access layer, such as JDBC.

3.5.4.5 Subsystem Architecture: Result Set

The Result Set is a wrapper around the JDBC ResultSet to provide transparent access to data stored in the result set. It allows the architecture to access the result set in an object-oriented manner. For example, sending next() to the result set will return the next business object instance



in the result set. Executing database operations such as select and detect on the Extent will return a Result Set of business object instances retrieved from the data store.

3.5.4.6 Subsystem Architecture: Business Mapper

The Business Mapper needs to provide the Business Object – specific mapping information to the rest of the framework. The ISFAPersistableMapper interface defines what methods will be required for a functional persistable business object mapper to implement. Implementing the interface for a particular business object will lead to a Mapper class, where all database specific code (select, insert, update, delete statements, etc.) will be contained. The framework then uses this code to map the business object to the database.

3.5.4.7 Subsystem Architecture: Business Object

The Business Object is a basic data structure with very little code overhead other than private data members and public set() and get() methods to access them. In this way the data that is retrieved from the database is completely decoupled from its Java representation. As a result once data is retrieved it can be manipulated with very little overhead (i.e. database access or connections) until it needs to be persisted back to the data store.



3.5.5 Detailed System Design

3.5.5.1 Component Definitions

ISFAPersistableMapper Interface

Interface Name:	ISFAPersistableMapper
Component:	Persistence
Description:	The ISFAPersistableMapper interface defines what methods are required for a functional persistable business object mapper.
Package:	gov.ed.sfa.ita.persistence
Superclass:	none

Attribute	Type	Description
Private:		
none		
Protected:		
none		
Public:		
none		

Con/Destructor	Arguments (Type, Name)	Description
none		

Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private:			
none			
Protected:			
none			
Public:			
getDeleteQuery	none	java.lang.String	This returns a DELETE query statement for the business object.
getInsertQuery	none	java.lang.String	This returns an INSERT query statement for the business object.
getKeySelectQuery	none	java.lang.String	This returns a SELECT query statement for the business object based on the key field.
getSelectQuery	java.lang.String selectCondition	java.lang.String	This returns a SELECT query statement for the business object.



Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
getUpdateQuery	none	java.lang.String	This returns a UPDATE query statement for the business object.
newFrom	gov.ed.sfa.ita.SFAResultSet sourceValues	java.lang.Object	This populates mapped attributes with the values from the result set.
populateAttributeValues	java.lang.Object businessObject	void	This method will populate all of the necessary attribute values from the business object in order to do an update or insert.
populateKeyAttributeValues	java.lang.Object businessObject	void	This method will populate all of the necessary attribute values from the business object in order to do a keyed select or delete.



SFAPersistableObjectManager Class

Class Name:	SFAPersistableObjectManagerClass
Component:	Persistence
Description:	The SFAPersistableObjectManager Class is used to create, read, update, and delete information from the data store. It creates (if necessary) and holds a single UnitOfWork that can be used throughout a Business Component method that accesses the data store.
Package:	gov.ed.sfa.ita.persistence
Superclass:	

Attribute	Type	Description
Private:		
transaction	SFAUnitOfWork	Used to hold a reference to the UnitOfWork.
databaseType	int	Used to track the current database type.
Protected:		
none		
Public:		
none		

Con/Destructor	Arguments (Type, Name)	Description
SFAPersistableObjectManager	SFADomain domain	This constructor takes a Domain and instantiates a UnitOfWork based on it.
SFAPersistableObjectManager	SFAUnitOfWork uow	This constructor uses the UnitOfWork given to it for its transactions.

Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private:			
none			
Protected:			
none			
Public:			
abortTransaction	none	void	Rolls back the current transaction.
addObject	java.lang.String insertString, java.util.Vector insertParams	void	Adds an object to the object store.
commitTransaction	none	void	Commits the current transaction.
endTransaction	none	void	Ends the current transaction



Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
getJDBCConnection	none	java.sql.Connection	Returns a handle to the JDBC Connection object that the POM is using.
getNextSequenceValue	java.lang.String sequenceName	java.lang.Integer	Gets an Integer value for the next sequence key value.
getObject	ISFAPersistableMapper persistMapper, java.lang.String.selectCondition, java.util.Vector retrieveParameters	java.lang.Object	Selects and returns the first object from the object store that meets the criteria supplied.
getObjects	ISFAPersistableMapper persistMapper, java.lang.String.selectCondition, java.util.Vector retrieveParameters	java.util.Vector	Retrieves a vector of objects from the object store matching the selectCondition.
getObjectsAsHashtable	ISFAPersistableMapper persistMapper, java.lang.String.selectCondition, java.util.Vector retrieveParameters	java.util.Hashtable	Retrieves a hashtable of objects from the object store matching the selectCondition.
removeObject	java.lang.String removeString, java.util.Vector removeParameters	void	Removes an object from the object store that meets the criteria supplied.
removeObjects	java.lang.String removeString, java.util.Vector removeParameters	void	Removes the objects from the object store that meet the criteria supplied.
setDatabaseType	int type	void	Sets the internal flag indicating the database type.
updateObject	java.lang.String updateCriteria, java.util.Vector updateParameters	void	Updates an object in the object store that meets the criteria supplied.
updateObjects	java.lang.String updateCriteria, java.util.Vector updateParameters	void	Updates objects in the object store that meet the criteria supplied.



SFAUnitOfWork Class

Class Name:	SFAUnitOfWork
Component:	Persistence
Description:	The SFAUnitOfWork class models a persistence unit of work. It has a Connection that represents the actual physical connection to the database. It allows a series of related actions to be logically grouped together that can be either committed or rolled back against a database.
Package:	gov.ed.sfa.ita.persistence
Superclass:	none

Attribute	Type	Description
Private:		
connection	java.sql.Connection	Holds the connection to the database.
Protected:		
none		
Public:		
none		

Con/Destructor	Arguments (Type, Name)	Description
SFAUnitOfWork	none	Default constructor.
SFAUnitOfWork	java.lang.String url, java.util.Properties dbInfo	Constructor with information on how to connect to the underlying database.

Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private:			
none			
Protected:			
none			
Public:			
abort	none	void	Rolls back all modifications made to the object store since the last commit, rollback, or checkpoint.
checkpoint	none	void	Commits all modifications made to the object store since the last commit, rollback, or checkpoint, and sets a checkpoint for rollbacks.



Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
commit	none	void	Commits all modifications made to the object store since the last commit, rollback or checkpoint.
end	none	void	Ends the unit of work and drops the connection to the object store domain (but not physically to the DBMS).
executeQuery	java.lang.String query	SFAResultSet	Takes the SQL query and executes it against the data store, returning an SFAResultSet.
executeUpdate	java.lang.String query	int	Executes a simple SQL update that returns a count of the number of rows updated.
getConnection	none	java.sql.Connection	Returns a Connection to the current data store.



SFAResultSet

Class Name:	SFAResultSet
Component:	Persistence
Description:	The SFAResultSet class is a wrapper around the JDBC ResultSet to provide transparent access to data stored in the result set. It allows the architecture to access the result set in an object-oriented manner.
Package:	gov.ed.sfa.ita.persistence
Superclass:	none

Attribute	Type	Description
Private:		
recordSet	java.sql.ResultSet	Holds the underlying result set.
Protected:		
none		
Public:		
none		

Con/Destructor	Arguments (Type, Name)	Description
SFAResultSet	java.sql.ResultSet aRecordSet	Creates an SFAResultSet based on the JDBC ResultSet passed to it.

Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private:			
none			
Protected:			
none			
Public:			
close	none	void	Closes the result set.
getBoolean	java.lang.String columnName	boolean	Returns the boolean value for the given column.
getDate	java.lang.String columnName	java.lang.Date	Returns the date value for the given column.
getFloat	java.lang.String columnName	float	Returns the float value for the given column.
getInt	java.lang.String columnName	int	Returns the int value for the given column.
getLong	java.lang.String columnName	long	Returns the long value for the given column.



Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
getObject	java.lang.String columnName	java.lang.Object	Returns the Object for the given column.
getString	java.lang.String columnName	java.lang.String	Returns the string value for the given column.
next	none	boolean	Moves to the next object in the SFAResultSet.



SFADomain Class

Class Name:	SFADomain
Component:	Persistence
Description:	The SFADomain class is a very simple class that is used by other classes. The Domain class acts as a holder for the connection string information, user name, password, and data source name. The SFADomain class contains no application logic.
Package:	gov.ed.sfa.ita.persistence
Superclass:	none

Attribute	Type	Description
Private:		
properties	java.util.Properties	Used to hold the user name and password.
connectionString	String	Used to hold the connection information.
USER	String	A constant string to act as the key for the user name in the properties object.
PASSWORD	String	A constant string to act as the key for the password in the properties object.
Protected:		
none		
Public:		
none		

Con/Destructor	Arguments (Type, Name)	Description
SFADomain		Default constructor.
SFADomain	java.util.Properties dbProperties	Constructor that takes a database properties object.
SFADomain	java.lang.String url	Constructor that takes a connection string.
SFADomain	java.lang.String connectionString, java.lang.String user, java.lang.String password	Constructor that takes a connection string, user name, and password.

Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private:			
none			
Protected:			
none			
Public:			



Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
getConnectionString	none	java.lang.String	Returns the connection string for the object.
getProperties	none	java.util.Properties	Returns the properties for the object.
setConnectionString	java.lang.String url	void	Sets the connection string.
setPassword	java.lang.String password	void	Sets the password.
setProperties	java.util.Properties dbProperties	void	Sets the properties.
setUser	java.lang.String user	void	Sets the user id.



SFAParser Class

Class Name:	SFAParser
Component:	Persistence
Description:	The SFAParser class converts parameterized strings into formatted SQL statements. This is done by first passing a parameterized string to a constructor, then setting the values of all the parameters, and then getting the result with the substituted parameters.
Package:	gov.ed.sfa.ita.persistence
Superclass:	

Attribute	Type	Description
Private:		
qParser	java.util.StringTokenizer	Used to parse through the statement.
Protected:		
parameterizedStatement	java.util.String	Holds the parameterized statement.
statementWithoutEscapes	java.util.String	Holds parameterized statement after intermediate processing step (escape characters removed).
m_parameters	java.util.Vector	Holds the values to be substituted for the parameters.
Public:		
none		

Con/Destructor	Arguments (Type, Name)	Description
SFAParser	java.lang.String statementString	Only constructor for the class - you must supply the statement string when instantiating this class.

Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private:			
none			
Protected:			
getEscapeChar	none	java.lang.String	
getNextToken	none	java.lang.String	
getParamDelimiter	none	java.lang.String	
getParser	none	java.util.StringTokenizer	
processRawStatement	none	void	
Public:			
getBooleanWrappedValue	SFAParameter q	java.lang.String	
getDateWrappedValue	SFAParameter q	java.lang.String	
getNumericWrappedValue	SFAParameter q	java.lang.String	



Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
getParameter	int i	SFAParameter	
getRawStatement	none	java.lang.String	
getStatement	none	java.lang.String	
getStringWrappedValue	SFAParameter q	java.lang.String	
getUserDefinedWrappedValue	SFAParameter q	java.lang.String	
getWrappedValue	SFAParameter q	java.lang.String	
setParams	java.util.Vector parameters	void	



SFAQueryParser Class

Class Name:	SFAQueryParser
Component:	Persistence
Description:	The SFAQueryParser is essentially the SFAParser class with two methods overridden. The class converts parameterized strings into formatted SQL statements. This is done by first passing a parameterized string to a constructor, then setting the values of all the parameters, and then getting the result with the substituted parameters.
Package:	gov.ed.sfa.ita.persistence
Superclass:	SFAParser

Attribute	Type	Description
Private:		
none		
Protected:		
none		
Public:		
none		

Con/Destructor	Arguments (Type, Name)	Description
SFAQueryParser	java.lang.String queryString	Only constructor for the class - you must supply the statement string when instantiating this class.

Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private:			
none			
Protected:			
processRawStatement	none	void	
Public:			
setParams	java.util.Vector parameters	void	



SFAOracleParser Class

Class Name:	SFAOracleParser
Component:	Persistence
Description:	The SFAOracleParser is essentially the SFAQueryParser class with two methods overridden. The class converts parameterized strings into formatted SQL statements. This is done by first passing a parameterized string to a constructor, then setting the values of all the parameters, and then getting the result with the substituted parameters.
Package:	gov.ed.sfa.ita.persistence
Superclass:	SFAParser

Attribute	Type	Description
Private:		
none		
Protected:		
none		
Public:		
none		

Con/Destructor	Arguments (Type, Name)	Description
SFAOracleParser	java.lang.String queryString	Only constructor for the class - you must supply the statement string when instantiating this class.

Method	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private:			
none			
Protected:			
none			
Public:			
getBooleanWrappedValue	SFAParameter q	java.lang.String	
getDateWrappedValue	SFAParameter q	java.lang.String	



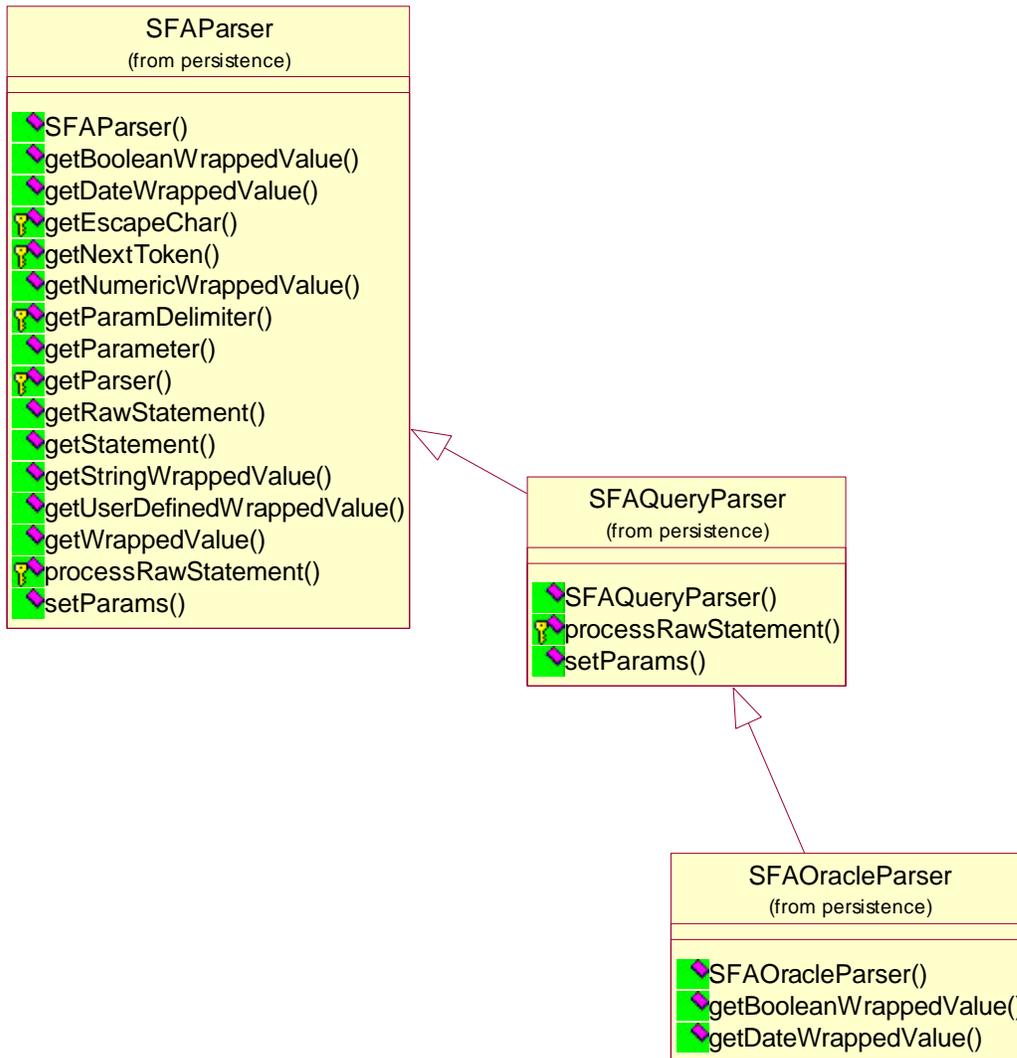
3.5.6 Class Diagrams

3.5.6.1 Persistence Classes





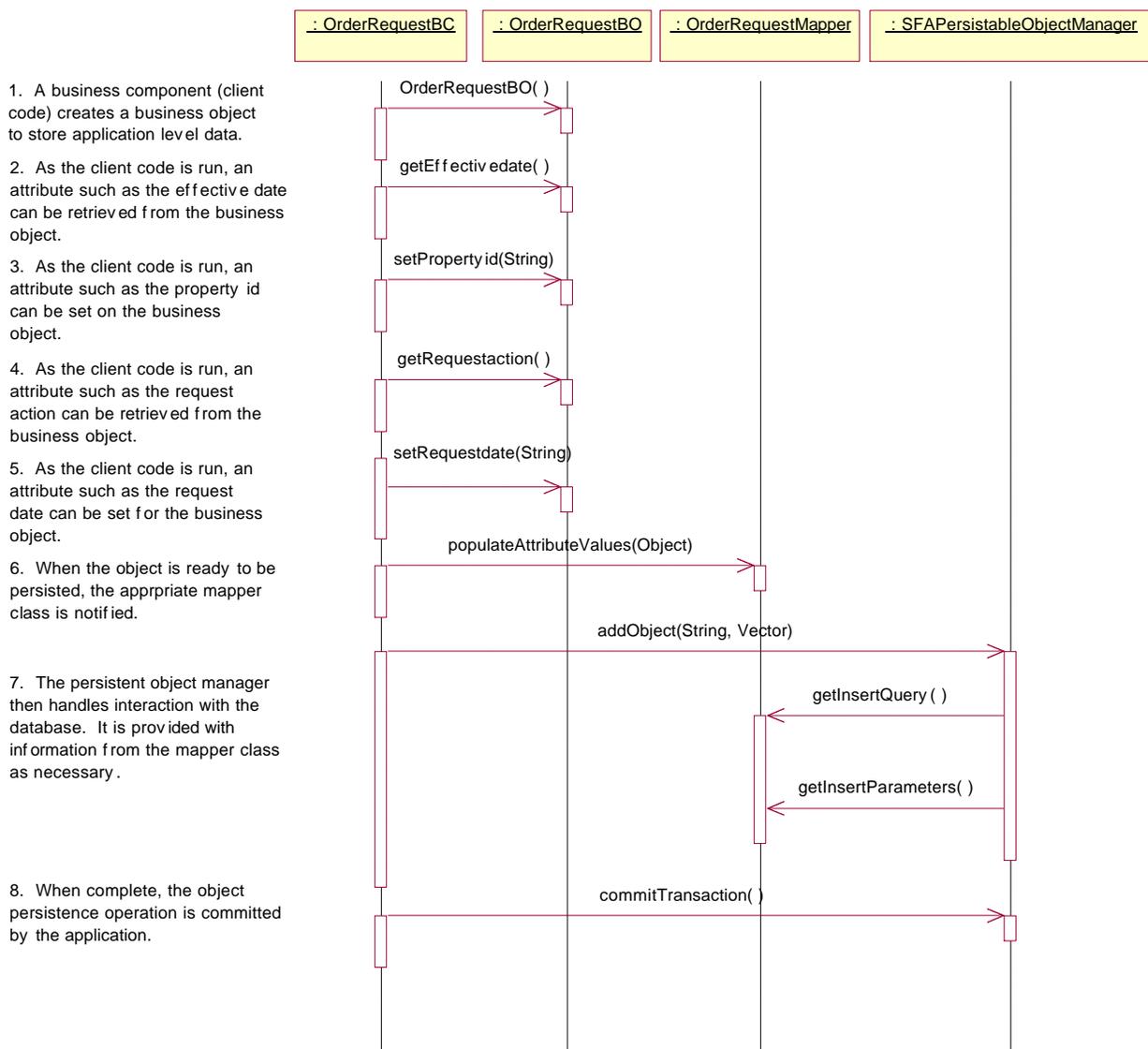
3.5.6.2 Parser Classes





3.5.7 Interaction Diagrams

This sequence diagram illustrates the main object interactions involved in persisting a new object to the database, from the perspective of client code. As can be seen from the diagram, a new business object is created, attributes are populated, and it is handed to the object manager, with the mapper providing object to database translation logic. Updates and deletes of persistable objects work in basically the same manner, so an understanding of the insert process, including the flow through to the mapper and persistable object manager, should serve as solid basis for all other persistence-type calls in this package.





3.5.8 References

- JavaDoc on JDBC
<http://www.javasoft.com/products/jdk/1.2/docs/api/index.html>
- JDBC Tutorial
<http://www.javasoft.com/docs/books/tutorial/jdbc/index.html>



3.6 Search

3.6.1 Introduction

The search framework simplifies, standardizes, and improves the use of the Autonomy search engine. The Autonomy Search Engine server provides a variety of methods to utilize its search capabilities including a C API, HTTP API, and configurable CGI program. Of the three ITA R1.0 applications that used Autonomy, IFAP and Intranet 2.0 used the C API to write a custom CGI program and Schools Portal used the configurable CGI program. Neither of these applications followed the J2EE standards of the ITA environment nor did the applications utilize all of the features provided by the Autonomy Search Engine Server.

This document covers only components that directly compose the Search framework. Consult the Autonomy Application Builder HTTP API documentation for more information on topics outside ones directly involved in the Search framework.

3.6.2 System Overview

The search framework complies with J2EE standards instead of using CGI as in the current search engine interface. The framework consists of classes that provide a common way to access the Autonomy HTTP API and utilize its features.

The search framework implements the following Autonomy features:

- Query search engine (including querying a custom field)
- Natural Language or "Fuzzy" query search engine (including querying a custom field)
- Display search results (including sorting by a custom field)
- Suggest additional search results

3.6.3 Design Considerations

3.6.3.1 Assumptions and Dependencies

It is assumed that this framework will be deployed in a J2EE application server environment. As the current production server for SFA is IBM WebSphere 3.5, the framework will be compiled using its required JDK version 1.2.2. It will also work with the current JavaServer Pages (1.1), Java Servlet (2.2), Java Messaging Service (1.0.1), and Java Database Connectivity (2.0) specifications for this server.

3.6.3.2 Goals and Guidelines

The goal of the search framework is to create a simplified and reusable Java based search framework for use with the Autonomy Search Engine server. This search framework standardizes the mechanism in which SFA applications access the Autonomy Search Engine



server. This search framework follows J2EE standards and is intended for use in JSPs, servlets, EJBs, or JavaBeans executed on the WebSphere Application Server. This search framework is also independent of Autonomy version and will allow for future upgrades of the Autonomy Search Engine server without modifications to this framework.

3.6.3.3 Development Methods

This framework was developed using general object-oriented software development techniques as are specified in any standard text on the Java programming language. As the framework itself is fairly straightforward in its class and relationship patterns it employs, no object oriented modeling tool or methodology was specifically used in its design. However, the resulting source code has been documented with standard class diagrams and sequence diagrams in order to illustrate its structure more readily. These diagrams should be of great help to programmers unfamiliar with this framework.

3.6.3.4 Future Autonomy Product Upgrades

An important design consideration for this search framework was compatibility for future Autonomy product upgrade. This design achieves compatibility for future upgrades of Autonomy on two fronts. Firstly, the design is based on the HTTP API, which has not changed since the first version of Autonomy. In addition, according to Autonomy technical representatives, the HTTP API will remain the method of communication and commands for future versions of Autonomy. Essentially the HTTP API is as fundamental to Autonomy communication as HTTP is to Web browsers and Web servers. Secondly, the design is flexible so that any new functionality or changes in functionality will not require any changes in any existing client code that uses this search framework. This is accomplished by not revealing any of the implementation details to the client using the search framework. The methods revealed to the client represent search functionality, but not any Autonomy Search Engine implementation details. For any additional features or changes in the Autonomy Search Engine product, these modifications will be made in the search framework and not in the client code. Together these two design considerations will help make this framework compatible with future versions of Autonomy products.

3.6.4 System Architecture

3.6.4.1 Overview

The Autonomy Search Engine server provides the capability for general queries, natural language or "fuzzy" queries, and suggestions of similar documents for Web-based applications. General queries allow for the traditional search capability within documents in addition to the ability to restrict search results to only those documents with specified field values or ranges or with similarity scores above a set threshold. Natural language or "fuzzy" queries score words and return results similar to the query in case the query language is slightly different than the terminology contained in the actual documents. Once the user has found a result that is relevant



to the original query, the Autonomy Search Engine server can suggest other documents similar to it.

Since there is significant similarity between Java to Autonomy Search Engine server interactions and Java to RDBMS database interactions, this search framework is based on the Java 2 Java Data Base Connectivity (JDBC) model for Connections, Statements, and ResultSets. This is a proven model that is familiar to most Java programmers making this search framework easy to learn and use.

The Search framework consists of three classes listed below:

- AutonomyConnection
- AutonomyStatement
- AutonomyRestultSet

These classes wrap the underlying HTTP API commands provided by the Autonomy Search Engine server. The AutonomyConnection class represents the Autonomy Search Engine Server connection information. The AutonomyStatement class represents statements sent to the Autonomy Search Engine server. The AutonomyResultSet class represents results returned from the Autonomy Search Engine server.

For exception handling and logging in this search framework, the RCS exception handling and logging frameworks will be used. These frameworks will ensure that sufficient exception handling and logging are included with this search framework.

Figure 8 below illustrates the interaction between any J2EE component (JSP, Servlet, EJB, JavaBean), the Search framework, and the Autonomy Search Engine server.

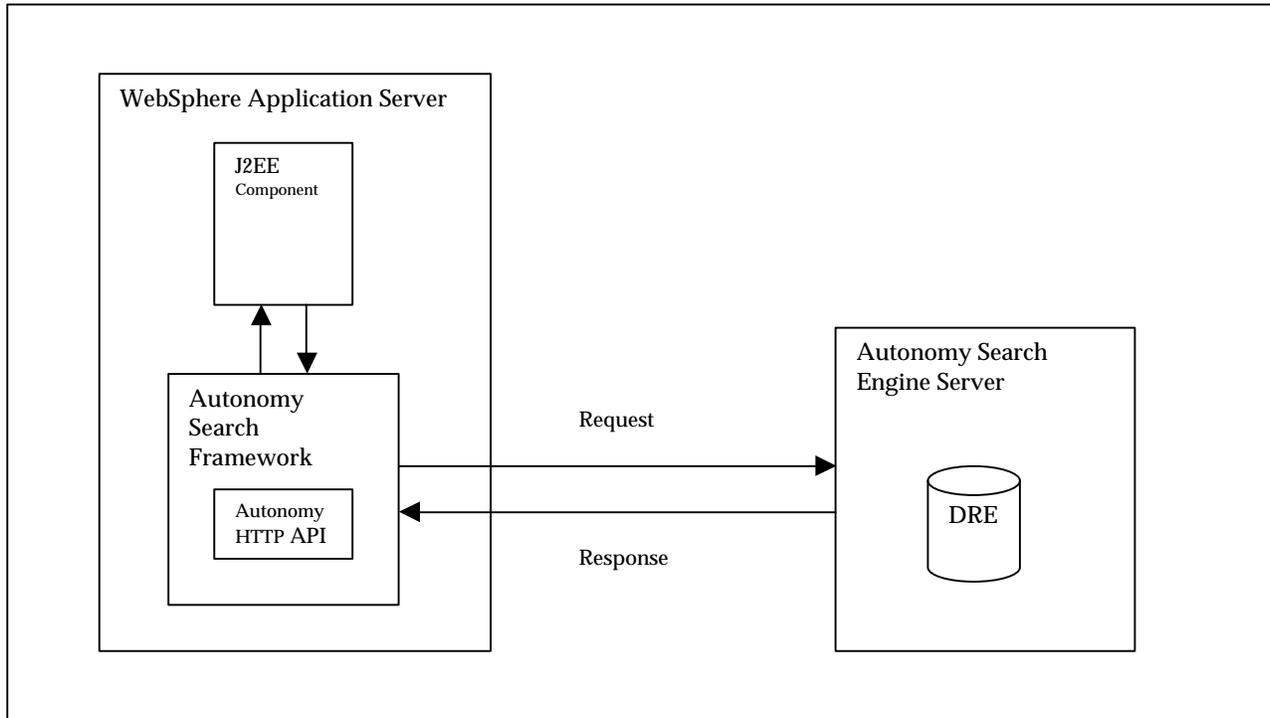


Figure 8: Interaction between Autonomy Search framework and Autonomy Search Engine Server

3.6.4.2 Subsystem Architecture: AutonomyConnection

This component represents the Autonomy Search Engine server. It contains the information necessary to connect to the Autonomy Search Engine server. It also contains a method to check the status of the Autonomy Search Engine server. This method uses an Autonomy HTTP API command and the java.net package to issue and process the HTTP command for status information from the Autonomy Search Engine Server. The Autonomy Search Engine server information such as the hostname and port numbers can vary and therefore this is read from an autonomy.properties file when creating the AutonomyConnection object. The createStatement method creates an AutonomyStatement object and with the proper connection information.

3.6.4.3 Subsystem Architecture: AutonomyStatement

This component represents statements sent to the Autonomy Search Engine server and follows the model of the Java 2 JDBC Statement class. Statements are created dynamically using the various methods contained in the AutonomyStatement class. These methods create the HTTP command string sent to the Autonomy Search Engine server. There are three methods that connect to the Autonomy Search Engine server and execute general queries, "fuzzy" queries, and suggestions based on similar documents. These three methods use the java.net package to issue



and process HTTP communication and commands to the Autonomy Search Engine server. There is a default query that is executed when all of the attributes of an `AutonomyStatement` are not set. The parameters of this default query are defined in an `autonomy.properties` file and are set at the creation of the `AutonomyStatement` object. The `AutonomyStatement` object can also be reset to these default parameters by using the `reset()` method.

For example, a statement is first created using the `createStatement` method on an `AutonomyConnection` object. The `AutonomyStatement` is then built dynamically using at minimum the `setQueryText()` and `setMaxNumResults()` methods. All other statement methods are optional unless the query needs to override the default search parameters in `autonomy.properties`. The statement is then sent to the Autonomy Search Engine server by calling `executeQuery()`. This method returns the correct number of results as an `AutonomyResultSet` object.

3.6.4.4 Subsystem Architecture: AutonomyResultSet

This component represents results returned from the Autonomy Search Engine server and follows the model of the Java 2 JDBC `ResultSet` class. `ResultSets` represent rows of search results with each row containing information such as the document title, document URL, and document summary. The two main types of methods inside `AutonomyResultSet` include methods for the iteration through the rows of the `AutonomyResultSet` and methods for getting information about each row. For example, the `next()` method sets the cursor to the next row in the `AutonomyResultSet` and returns true as long as there are more rows left. An example of the other type of method includes the `getTitle()` method, which returns the document title of the current row's result.

The `AutonomyResultSet` object is returned after an `executeQuery`, `executeFuzzyQuery`, or `executeSuggest` method is invoked on an `AutonomyStatement` object. Internal to the `AutonomyResultSet` class, the actual results from the Autonomy Search Engine server are pure text and are contained in an `InputStream` object that is then converted to a `BufferedReader` object for easier parsing and text manipulation.



3.6.5 Detailed System Design

AutonomyConnection

Class Name:	AutonomyConnection
Component:	Search
Description:	This class represents the Autonomy search server.
Package:	gov.ed.sfa.ita.search
Superclass:	none

Attribute	Type	Description
Private:		
host	String	Host name of Autonomy search server
indexPort	int	Index port number of Autonomy search server
queryPort	int	Query port number of Autonomy search server
Public:		
none		

Con/Destructors	Arguments	Description
	(Type, Name)	
AutonomyConnection	None	Creates Autonomy object and assigns host, queryPort, and indexPort according to a autonomy.properties file

Methods	Arguments	Valid Responses	Description
	(Type, Name)	(Return Type, Exceptions Thrown)	



Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private			
getProperties	None	boolean	Reads the Autonomy.properties file and sets attributes
Public:			
createStatement	None	AutonomyStatement	Returns an AutonomyStatement object with the proper connection information
getHost	None	String	Returns the hostname of Autonomy object
getIndexPort	None	int	Returns the index port of Autonomy object
getQueryPort	None	int	Returns the query port of Autonomy object
isServerStatusOK	None	boolean	Returns true if Autonomy server status indicates the server is running



AutonomyStatement

Class Name:	AutonomyStatement
Component:	Search
Description:	This class represents statements sent the Autonomy search server such as search queries.
Package:	gov.ed.sfa.ita.search
Superclass:	None

Attribute	Type	Description
Private:		
aCon	AutonomyConnection	Reference to Autonomy Connection object
aRS	AutonomyResultSet	Reference to AutonomyResultSet object
attachtoname	String	Represents list of database names to query within Autonomy
attachtonum	String	Represents list of database numbers to query within Autonomy
customField	String	Represents any custom or extra options added by a user to the Autonomy search server or upgrades
docid	String	Represents a list of document ids
qmethod	String	Indicates type of query to perform
qnum	int	Represents maximum number of results to return
querystmt	String	Represents full query string to send
querytext	String	Represents list of search terms
threshold	int	Represent minimum threshold of results to return
xoptions	String	Represents extra options for searches such as sort by date



Con/Destructors	Arguments (Type, Name)	Description
AutonomyStatement	AutonomyConnection aCon	Creates AutonomyStatement object and uses the AutonomyConnection object

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private:			
getProperties	None	boolean	Reads the Autonomy.properties file and sets attributes
Public:			
execute	String querystmt	AutonomyResultSet throw AutonomyException	Executes a generic query statement passed as parameter. Called by other execute methods to do all communication
executeFuzzyQuery	none	AutonomyResultSet throws AutonomyException	Executes a fuzzy query on the Autonomy search server
executeQuery	none	AutonomyResultSet throws AutonomyException	Executes a query on the Autonomy search server
executeSuggest	String docid	AutonomyResultSet throws SFAException	Executes a suggest on the Autonomy search server based on the selected document
setCustomField	String fieldName	none	Sets the values of the optional custom field
setDatabaseNums	String attachto	none	Sets the list of databases to query in Autonomy
setDatabaseNames	String attachtoname	none	Sets the list of databases to query in Autonomy



Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
setMaxNumResults	int qnum	none	Sets the maximum number of results
setMinThreshold	int threshold	none	Sets the minimum threshold for results
setQueryAllWords	none	none	Sets the query to a Boolean AND
setQueryAnyWords	none	none	Sets the query to a Boolean OR
setQueryText	String querytext	none	Sets the query terms for searches
setQueryTextCustomField	String querytext, String fieldname	none	Set the query terms fir searches in a particular field
setSortByCustomField	String fieldName	none	Sets the option to return results by sorting based on the custom field
setSortByDate	none	none	Sets the option to return results by date and not relevance
setSortByRelevance	none	none	Sets the option to return results by relevance and not by date
setSortByRelevanceDate	none	none	Sets the option to return results by relevance and then date
reset	none	boolean	Resets the object to the default search parameters in autonomy.properties file



AutonomyResultSet

Class Name:	AutonomyResultSet
Component:	Search
Description:	This class represents results returned from the Autonomy search server
Package:	gov.ed.sfa.ita.search
Superclass:	None

Attribute	Type	Description
Private:		
bfrResults	BufferedReader	Buffered reader results used for parsing Autonomy query raw results
currentRow	int	Current row location of cursor
istrResults	InputStream	The raw results returned from the Autonomy query
numRows	int	Number of rows in AutonomyResultSet
Public:		
none		

Con/Destructors	Arguments	Description
	(Type, Name)	
AutonomyResultSet	InputStream istrResults	Creates AutonomyResultSet object with InputStream as parameter

Methods	Arguments	Valid Responses	Description
	(Type, Name)	(Return Type, Exceptions Thrown)	



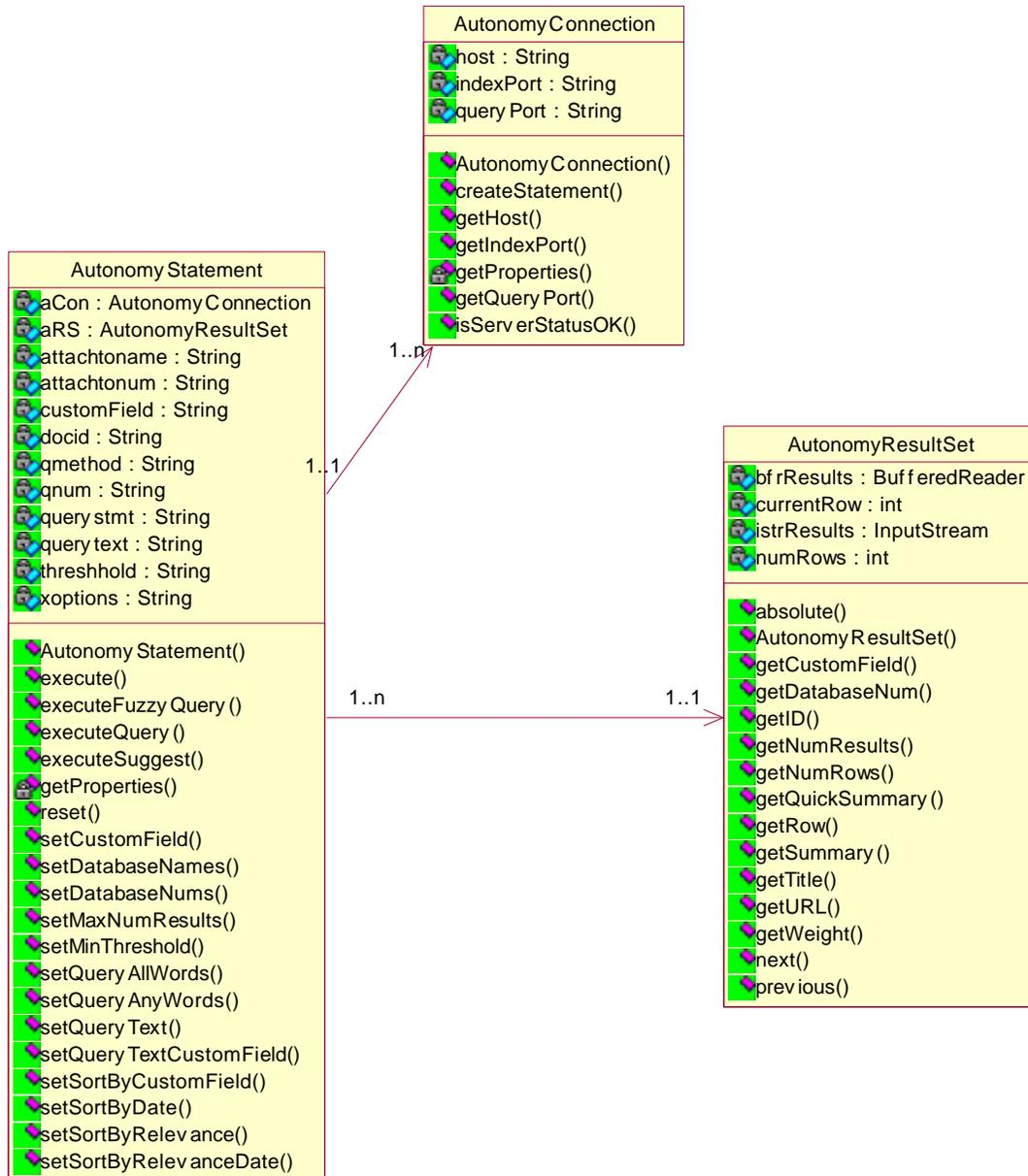
Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
absolute	int row	boolean	Jumps to the row specified in parameter, returns true is successful
getCustomField	String fieldName	String	Returns the value of the custom field defined in the parameter
getDatabaseNum	none	int	Returns document database number of current row
getID	none	String	Returns document id of current row
getNumResults	none	int	Returns the number of results returned in the query
getNumRows	none	int	Returns the number of rows in the AutonomyResultSet
getQuickSummary	none	String	Returns the quick summary of the current row
getRow	none	int	Returns the current row location of the cursor
getSummary	none	String	Returns document summary of current row
getTitle	none	String	Returns document title of current row
getURL	none	String	Returns document URL of current row
getWeight	none	int	Returns document weight of current row
next	none	boolean	Moves to the next row in the AutonomyResultSet, returns true if successful



Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
previous	none	boolean	Moves to the previous row in the AutonomyResultSet, returns true if successful



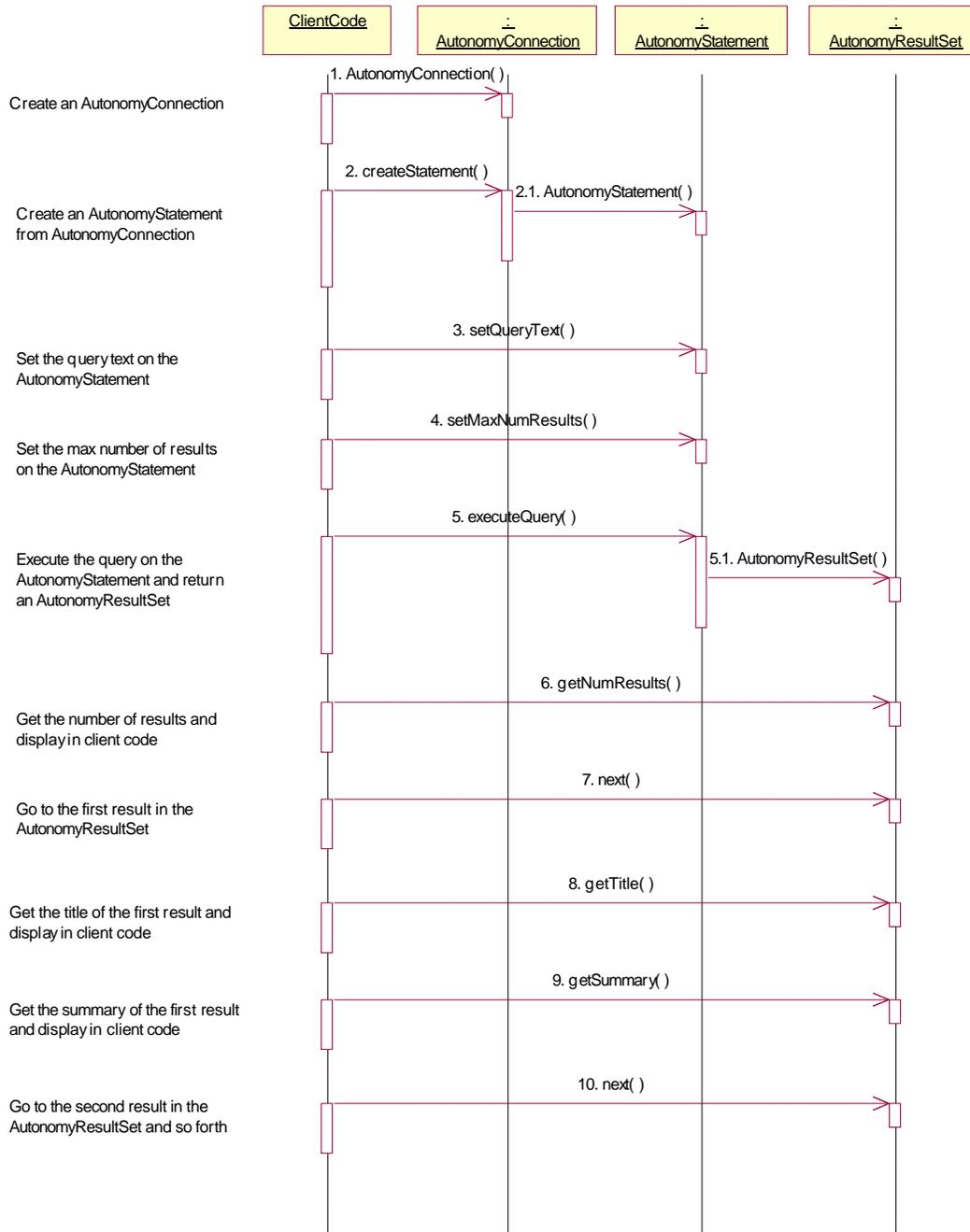
3.6.6 Class Diagrams





3.6.7 Interaction Diagrams

The following is an interaction diagram for executing a search on the Autonomy Search Engine server and displaying the total number of results and the title and summary of the first result.





3.6.8 References

- Autonomy Search Engine Server web-site
<http://www.autonomy.com> (follow links to "Solutions->Autonomy Server")
- Autonomy HTTP API Documentation:
See document entitled "Autonomy Application Builder: HTTP API" and located on the shared X: drive under X:\\CIO\\ITA2\\AUTONOMY DOCS\\



Appendix A – ITA RCS and SFA Applications Matrix

The following matrix details the current and potential usage of RCS services by SFA’s applications.

	<i>In Use</i>	<i>High Opportunity</i>	<i>Medium Opportunity</i>	<i>Low Opportunity</i>	<i>Not Applicable</i>
<i>Component Factory</i>			EIP	IFAP, Intranet R2.0, Schools Portal	CBS, FAFSA
<i>Email</i>	CBS (7/16)	Intranet R2.0, EIP, IFAP		Schools Portal	FAFSA
<i>Exception Handling</i>	FAFSA (6/20)	EIP	Intranet R2.0, IFAP, Schools Portal		CBS
<i>Logging</i>	FAFSA (6/20)	EIP	Intranet R2.0, IFAP, Schools Portal		CBS
<i>Persistence</i>	CBS (7/16)	EIP		IFAP	Intranet R2.0, Schools Portal, FAFSA
<i>Search</i>	Intranet R2.0 (8/3), IFAP (8/17), Schools Portal (8/31)	EIP			CBS, FAFSA

Last Updated: 6/28/01