Oracle Secure Enterprise Search
Secure Connector Software Development Kit

Oracle Secure Enterprise Search Version 10.1.6
June 2006

Introduction

Oracle Secure Enterprise Search 10.1.6 supports secure connectors out-of-the-box that enable crawling of documents in Web servers, database tables, IMAP servers, file systems, and Oracle Portal. However, many enterprises have their own proprietary document and data repositories. Some of these are created in-house while others are purchased from software vendors. The Oracle Secure Enterprise Search's Secure Connector Development Kit enables customers and vendors to develop their own Secure Connector to crawl the content of both proprietary and commercial document or data repository.

The Secure Connector Software Development Kit comprises of the following pieces of information to aid in developing a secure connector:

a. A Quick Start Guide to developing a Secure Crawler Plug-in that is part of this document.
b. Chapter 3 of the Administrator's Guide that outlines the Secure Crawler concepts. URL: http://download-west.oracle.com/docs/cd/B28527_01/doc/search.1016/b19002/toc.htm
c. A tutorial that teaches you how to develop the Secure Crawler Plug-in. URL: http://st-curriculum.oracle.com/tutorial/SESDevTutorial/index.htm
d. Java Doc that describes the interfaces to be implemented. URL: http://download-west.oracle.com/docs/cd/B28527_01/doc/search.1016/b19032/toc.htm
e. Sample Crawler Plug-in is also available to provide a jump start on developing your own secure connector. URL: http://www.oracle.com/technology/products/oses/files/xmlplugin.zip

Quick Start Guide for Creating Secure Crawler Plug-in

This section provides a quick overview of the Oracle Secure Enterprise Search's Secure Crawler plug-in creation process.

Steps to Create a Crawler Plug-in

---

1. **Confirm the need for the plug-in**
   The web application of the repository may be amenable to crawl as a web source. Check if the repository supports other access protocol such that a custom solution can be avoided. For example, LDAP.

2. **Check plug-in availability**
   Check if such a plug-in is already available from Oracle or third-party vendors. Oracle shipped three sample plug-ins that supports LDAP crawl, database table crawl, and file directory crawl.

3. **Study the target repository**
   Determine how to access the repository for document contents, metadata, and (optionally) access information. Is there single instance for such a repository or hundreds of them? This could affect the plug-in design. For example, should the plug-in crawl one repository at a time or all of them?

4. **Settle on the document model**
   Determine what constitutes a document. It could be a virtual document synthesized on the fly or could contain only metadata. Decide on the set of metadata that should be collected as document attributes that can be searched.

5. **Define URL for the target document**
   It could be a HTTP URL generated by the web interface application provided by the repository vendor or a URL pointing to an enterprise application in the context of the search hit. For example, a vacation report application. Note there could be two kinds of URL for a document. One is called display URL that is used by the end user in the search result; the other is called access URL that is internal to the plug-in for it to retrieve the document. The display URL must be unique within a data source.

6. **Decide how to access the repository**
   What kind of information is needed for accessing the repository? For a database it could be the database connect string and the schema name and password. Keep in mind the crawler is a stand-alone java process; it is not part of the ORACLE SECURE ENTERPRISE SEARCH oc4j web application.

7. **Decide on a data source model**
   A data source is a set of documents that can be treated as a logical unit. For example, a table from a database, a web site from the intranet, or a file directory. A data source is also the unit of crawling. The crawler crawls one data source at a time. For the target repository, there should be concrete parameters that can

define such a source.

8. **Provide document access control**
   Technically, this could be the most difficult step in creating a plug-in if the document is to be protected on a per-document basis. An Access Control List (ACL) needs to be assigned to a document crawled by Secure Enterprise Search. The principal, in the form of a global user id (GUID) or distinguished-name (DN), in the ACL must exist in the LDAP server that Secure Enterprise Search uses as its Identity Management Store. In Secure Enterprise Search 10.1.6 only Oracle Internet Directory (OID) is supported. If the repository in question uses an alternate LDAP server (e.g., Microsoft Active Directory) then the mapping of the users between the repository and OID must be maintained which is outside the scope of the plug-in. Note that Secure Enterprise Search supports data source level access control by allowing the administrator to stamp an ACL to the data source.

9. **Decide on how to crawl the data source**
   ”crawl” the data source means go through the list of documents in the data source and retrieve them for indexing. One important crawl design decision here is whether to use a queue (provided by plug-in API’s QueueService) or not. For example, if this is a table crawl, the table could be partitioned by the number of crawling threads (one plug-in per thread) and each plug-in makes a database connection and perform a SQL select. In that case each plug-in keeps fetching the document and submit directly to the crawler without going through the queue. On the other hand a file directory crawl or any tree-structured data source can conveniently use the queue for the crawl. Note that document contents cannot be put into the queue.

10. **Think about how to recrawl**
    The first time crawl is always retrieving all of the documents from the data source. But what should the plug-in do on subsequent crawl? Will the plug-in be able to detect the set of insert/update/delete documents since last crawl? In general detecting the deleted document set is not trivial. How does the plug-in know this is a recrawl and to behave differently?

11. **Leverage the sample plug-in**
    ORACLE SECURE ENTERPRISE SEARCH shipped with three sample plug-ins under $OH/search/sample/agent/. Study the sample get a sense of what the plug-in looks like.

12. **Build the plug-in java classes**
    Implementing the Oracle Secure Enterprise Search java Crawler plug-in API inetrface *CrawlerPluginManager* and *CrawlerPlugin* creates a Secure Crawler plug-in. Oracle Secure Enterprise Search crawler implements the rest of the plug-in APIs. Clearly defining a list of parameters is important for users of the plug-in to configure the plug-in.. For the list of APIs to use, please see the API summary below and the Secure Enterprise Search’s Javadoc documentation.

13. **Package the plug-in java classes into a jar file**
    A simple plug-in could package all its classes into a single plug-in jar file. If there are separate jar files the plug-in depends on, its class path should be specified in

the *Class-Path* of the plug-in jar file *MANIFEST.MF*. Put the jar file under $OH/search/lib/agent/ directory. If the plug-in code also relies on a particular library file (for example, a .dll file on Windows or a .so file on UNIX), then the library path environment variable (`PATH` on Windows, `LD_LIBRARY_PATH` on UNIX) must contain the path to it. Make sure that Oracle is started from this environment. As the Oracle process spawns the crawler, it automatically inherits all environment variables from Oracle, including the library path.

14. **Register the plug-in**
    The plug-in defines a user-defined data source type. Use the browser to go to the admin page, under <Global-Setting>/<Source Types>, click on the 'create' button and provide the java class name that implements *CrawlerPluginManager* and the corresponding plug-in jar file name. For example, "app.crawler.ldap.LdapCrawlerPluginManager" and "pluginLdap.jar

15. **Create a new data source based on the plug-in defined source type**
    Specify the target repository data source type when creating a new data source. Values for the plug-in parameters must be specified here. A crawler schedule will be created automatically for this new data source.

16. **Test the plug-in**
    Start the scheduled crawl and check for any plug-in problem from the crawler log file. This most likely will be an iteration process and requires revising the plug-in code and repackaging it. If there is no change to the plug-in parameters then there is no need to re-register the plug-in.

17. **Test searching the crawled data source**
    If the crawl finishes successfully, test querying the data source to see the search hit and verify the search hit URL is working correctly.

Crawler Plug-in API Summary

Interface Summary	
CrawlerPlugin	Implemented by plug-in writer crawl() method is the heart of the plug-in. key method: crawl( )
CrawlerPluginManager	Implemented by plug-in writer. Responsible for plug-in registration and materializing plug-in instance to be used by the crawler. key method: init(), getCrawlerPlugin(), getPluginParameters()
CrawlingThreadService	Entry point for submitting document to the crawler. key method: submitForProcessing(DocumentContainer

	target)
DataSourceService	An optional service used for managing data source key method: delete(url), indexNow(), registerGlobalLOV()
DocumentAcl	Object for holding document access control principal. Save it to DocumentMetadata object. key method: addPrincipal(), addDenyPrincipal()
DocumentContainer	A document “holder” for the document. Note metadata and document status must be set in order to submit the document. key method: setMetadata(DocumentMetadata), setDocument(InputStream), setDocument(Reader), setDocumentStatus()
DocumentMetadata	Object for storing document metadata and access control information. key method: setACLInfo(DocumentAcl), setAttribute(), setContentType(), setSourceHierarchy()
GeneralService	Entry point for getting DataSourceService, QueueService, and LoggingService. Factory for creating DocumentAcl, DocumentMetadata, DocumentContainer, and LovInfo object.
Logger	Logging interface to output message to the crawler log file. key method: error(), fatal(), info(), warn()
LovInfo	Object for holding search attribute list of values key method: addAttributeValue(name, value)
ParameterValues	An interface for the plug-in to read the value of data source parameter.
QueueService	An optional service for storing pending document URLs. key method: enqueue(), getNextItem()

Class Summary	
ParameterInfo	ParameterInfo is a class for describing the general properties of a parameter. PluginManager returns a list of ParameterInfo through

	getPluginParameters().

Exception Summary	
PluginException	An exception thrown by the plug-in to report error. This will shut down the crawler if isFatalException() is true.
ProcessingException	Exception thrown by the crawler to the plug-in to indicates trouble processing plug-in's request. If this is a fatal error the crawler will try to shut down. Otherwise it's up to the plug-in to continue to the next document or not.