lucidworks Enterprise

Built on Solr

Simplified, Accelerated Productivity

Cost Effective Architecture

For the Developer Access Release of LucidWorks Enterprise, go to

http://bit.ly/lucidworks

lucid IMAGINATION

The commercial entity for Apache Solr and Lucene.

Apache Solr

Lucene

lucidimagination.com

DZone Refcardz

brought to you by... lucid IMAGINATION

CONTENTS INCLUDE:

Apache Solr:
Getting Optimal Search Results

By Chris Hostetter

ABOUT SOLR

Solr makes it easy for programmers to develop sophisticated, high performance search applications with advanced features such as faceting, dynamic clustering, database integration and rich document handling.

Solr (http://lucene.apache.org/solr/) is the HTTP based server product of the Apache Lucene Project. It uses the Lucene Java library at its core for indexing and search technology, as well as spell checking, hit highlighting, and advanced analysis/tokenization capabilities.

The fundamental premise of Solr is simple. You feed it a lot of information, then later you can ask it questions and find the piece of information you want. Feeding in information is called indexing or updating. Asking a question is called a querying.

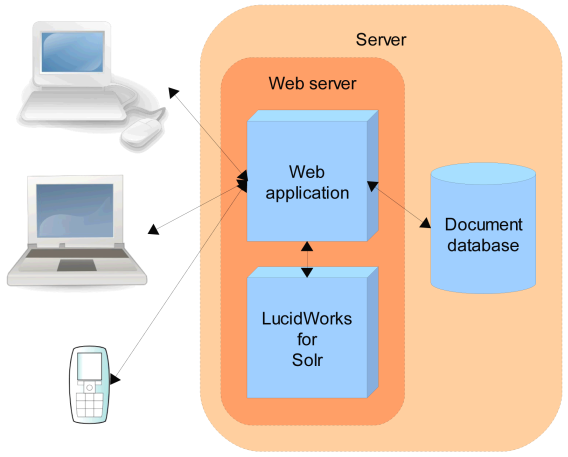


Figure 1: A typical Solr setup

Core Solr Concepts

Solr's basic unit of information is a document: a set of information that describes something, like a class in Java. Documents themselves are composed of fields. These are more specific pieces of information, like attributes in a class.

RUNNING SOLR

Solr Installation

The LucidWorks for Solr installer (http://www.lucidimagination.com/Downloads/LucidWorks-for-Solr) makes it easy to set up your initial Solr instance. The installer brings you through configuration and deployment of the Web service on either Jetty or Tomcat.

Solr Home Directory

Solr Home is the main directory where Solr will look for configuration files, data and plug-ins.

When LucidWorks is installed at `~/LucidWorks` the Solr Home directory is `~/LucidWorks/lucidworks/solr/`.

Single Core and Multicore Setup

By default, Solr is set up to manage a single "Solr Core" which contains one index. It is also possible to segment Solr into multiple virtual instances of cores, each with its own configuration and indices. Cores can be dedicated to a single application, or to different ones, but all are administered through a common administration interface.

Multiple Solr Cores can be configured by placing a file named `solr.xml` in your Solr Home directory, identifying each Solr Core, and the corresponding instance directory for each. When using a single Solr Core, the Solr Home directory is automatically the instance directory for your Solr Core.

Configuration of each Solr Core is done through two main config files, both of which are placed in the conf subdirectory for that Core:

- `schema.xml`: where you describe your data
- `solrconfig.xml`: where you describe how people can interact with your data.

By default, Solr will store the index inside the data subdirectory for that Core.

Solr Administration

Administration for Solr can be done through http://[hostname]:8983 /solr/admin which provides a section with menu items for monitoring indexing and performance statistics, information about index distribution and replication, and information on all threads running in the JVM at the time. There is also a section where you can run queries, and an assistance area.

SCHEMA.XML

To build a searchable index, Solr takes in documents composed of data fields of specific field types. The `schema.xml` configuration file defines the field types and specific fields that your documents can contain, as well as how Solr should handle those fields when adding documents to the index or when querying those fields. When you perform a query, `schema.xml` is structured as follows:

```
<schema>
   <types>
   <fields>
   <uniqueKey>
   <defaultSearchField>
   <solrQueryParser>
   <copyField>
</schema>
```

FIELD TYPES

A field type includes three important pieces of information:

- The `name` of the field type
- Implementation `class` name
- Field attributes

Field types are defined in the types element of schema.xml.

```
<fieldType name="textTight" class="solr.TextField">
…
</fieldType>
```

The type name is specified in the `name` attribute of the `fieldType` element. The name of the implementing class, which makes sure the field is handled correctly, is referenced using the `class` attribute.

Hot Tip

Shorthand for Class References
When referencing classes in Solr, the string `solr` is used as shorthand in place of full Solr package names, such as `org.apache.solr.schema` or `org.apache.solr.analysis`.

Numeric Types

Solr supports two distinct groups of field types for dealing with numeric data:

- Numerics with Trie Encoding: `TrieDateField`, `TrieDoubleField`, `TrieIntField`, `TrieFloatField`, and `TrieLongField`.
- Numerics Encoded As Strings: `DateField`, `SortableDoubleField`, `SortableIntField`, `SortableFloatField`, and `SortableLongField`.

Which Type to Use?

Trie encoded types support faster range queries, and sorting on these fields is more RAM efficient. Documents that do not have a value for a Trie field will be sorted as if they contained the value of "0". String encoded types are less efficient for range queries and sorting, but support the `sortMissingLast` and `sortMissingFirst` attributes.

Class	Description
`BinaryField`	Binary data that needs to be base64 encoded when reading or writing
`BoolField`	Contains either true or false. Values of "1", "t", or "T" in the first character are interpreted as true. Any other values in the first character are interpreted as false.
`ExternalFileField`	Pulls values from a file on disk.
`RandomSortField`	Does not contain a value. Queries that sort on this field type will return results in random order. Use a dynamic field to use this feature.
`StrField`	String
`TextField`	Text, usually multiple words or tokens
`UUIDField`	Universally Unique Identifier (UUID). Pass in a value of "NEW" and Solr will create a new UUID.

Hot Tip

Date Field
Dates are of the format `YYYY-MM-DDThh:mm:ssZ`. The Z is the timezone indicator (for UTC) in the canonical representation. Solr requires date and times to be in the canonical form, so clients are required to format and parse dates in UTC when dealing with Solr. Date fields also support date math, such as expressing a time two months from now using `NOW+2MONTHS`.

Field Type Properties

The field class determines most of the behavior of a field type, but optional properties can also be defined in `schema.xml`.

Some important Boolean properties are:

Property	Description
`indexed`	If `true`, the value of the field can be used in queries to retrieve matching documents. This is also required for fields where sorting is needed.
`stored`	If `true`, the actual value of the field can be retrieved in query results.
`sortMissingFirst` `sortMissingLast`	Control the placement of documents when a sort field is not present in supporting field types.
`multiValued`	If `true`, indicates that a single document might contain multiple values for this field type.

ANALYZERS

Field analyzers are used both during ingestion, when a document is indexed, and at query time. Analyzers are only valid for `<fieldType>` declarations that specify the `TextField` class. Analyzers may be a single class or they may be composed of a series of zero or more `CharFilter`, one `Tokenizer` and zero or more `TokenFilter` classes.

Analyzers are specified by adding `<analyzer>` children to the `<fieldType>` element in the `schema.xml` config file. Field Types typically use a single analyzer, but the `type` attribute can be used to specify distinct analyzers for the `index` vs `query`.

The simplest way to configure an analyzer is with a single `<analyzer>` element whose class attribute is the fully qualified Java class name of an existing Lucene analyzer.

For more configurable analysis, an analyzer chain can be created using a simple `<analyzer>` element with no class attribute, with the child elements that name factory classes for `CharFilter`, `Tokenizer` and `TokenFilter` to use, and in the order they should run, as in the following example:

```
<fieldType name="nametext" class="solr.TextField">
  <analyzer>
    <charFilter class="solr.HTMLStripCharFilterFactory"/>
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StandardFilterFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

CharFilter

`CharFilter` pre-process input characters with the possibility to add, remove or change characters while preserving the original character offsets.

The following table provides an overview of some of the `CharFilter` factories available in Solr 1.4:

CharFilter	Description
`MappingCharFilterFactory`	Applies mapping contained in a map to the character stream. The map contains pairings of String input to String output.
`PatternReplaceCharFilterFactory`	Applies a regular expression pattern to the string in the character stream, replacing matches with the specified replacement string.
`HTMLStripCharFilterFactory`	Strips HTML from the input stream and passes the result to either a CharFilter or a Tokenizer. This filter removes tags while keeping content. It also removes <script>, <style>, comments, and processing instructions.

Tokenizer

`Tokenizer` breaks up a stream of text into tokens. `Tokenizer` reads from a Reader and produces a `TokenStream` containing various metadata such as the locations at which each token occurs in the field.

The following table provides an overview of some of the `Tokenizer` factory classes included in Solr 1.4:

Tokenizer	Description
`StandardTokenizerFactory`	Treats whitespace and punctuation as delimiters.
`NGramTokenizerFactory`	Generates n-gram tokens of sizes in the given range.
`EdgeNGramTokenizerFactory`	Generates edge n-gram tokens of sizes in the given range.
`PatternTokenizerFactory`	Uses a Java regular expression to break the text stream into tokens.
`WhitespaceTokenizerFactory`	Splits the text stream on whitespace, returning sequences of non-whitespace characters as tokens.

TokenFilter

`TokenFilter` consumes and produces `TokenStreams`. `TokenFilter` looks at each token sequentially and decides to pass it along, replace it or discard it.

A `TokenFilter` may also do more complex analysis by buffering to look ahead and consider multiple tokens at once.

The following table provides an overview of some of the `TokenFilter` factory classes included in Solr 1.4:

TokenFilter	Description
`KeepWordFilterFactory`	Discards all tokens except those that are listed in the given word list. Inverse of `StopFilterFactory`.
`LengthFilterFactory`	Passes tokens whose length falls within the min/max limit specified.
`LowerCaseFilterFactory`	Converts any uppercases letters in a token to lowercase.
`PatternReplaceFilterFactory`	Applies a regular expression to each token, and substitutes the given replacement string in place of the matched pattern.
`PhoneticFilterFactory`	Creates tokens using one of the phonetic encoding algorithms from the `org.apache.commons.codec.language` package.
`PorterStemFilterFactory`	An algorithmic stemmer that is not as accurate as table-based stemmer, but faster and less complex.
`ShingleFilterFactory`	Constructs shingles (token n-grams) from the token stream.
`StandardFilterFactory`	Removes dots from acronyms and 's from the end of tokens. This class only works when used in conjunction with the `StandardTokenizerFactory`
`StopFilterFactory`	Discards, or stops, analysis of tokens that are on the given stop words list.
`SynonymFilterFactory`	Each token is looked up in the list of synonyms and if a match is found, then the synonym is emitted in place of the token.
`TrimFilterFactory`	Trims leading and trailing whitespace from tokens.
`WordDelimitedFilterFactory`	Splits and recombines tokens at punctuations, case change and numbers. Useful for indexing part numbers that have multiple variations in formatting.

Hot Tip

Testing Your Analyzer
There is a handy page in the Solr admin interface that allows you to test out your analysis against a field type at the http://[hostname]:8983/solr/admin/analysis.jsp page in your installation.

FIELDS

Once you have field types set up, defining the fields themselves is simple: all you need to do is supply the `name` and a reference to the name of the declared `type` you wish to use. You can also provide options that override the options for that field type.

```
<field name="price" type="sfloat" indexed="true"/>
```

Dynamic Fields

Dynamic fields allow you to define behavior for fields that are not explicitly defined in the schema, allowing you to have fields in your document whose underlying `<fieldType/>` will be driven by the field naming convention instead of having an explicit declaration for every field.

Dynamic fields are also defined in the fields element of the schema, and have a `name`, `field type`, and `options`.

```
<dynamicField name="*_i" type="sint" indexed="true" stored="true"/>
```

OTHER SCHEMA ELEMENTS

Copying Fields

Solr has a mechanism for making copies of fields so that you can apply several distinct field types to a single piece of incoming information.

```
<copyField source="cat" dest="text" maxChars="30000" />
```

Unique Key

The `uniqueKey` element specifies which field is a unique identifier for documents. Although `uniqueKey` is not required, it is nearly always warranted by your application design. For example, `uniqueKey` should be used if you will ever update a document in the index.

```
<uniqueKey>id</uniqueKey>
```

Default Search Field

If you are using the Lucene query parser, queries that don't specify a field name will use the `defaultSearchField`. The dismax query parser does not use this value in Solr 1.4.

```
<defaultSearchField>text</defaultSearchField>
```

Query Parser Operator

In queries with multiple clauses that are not explicitly required or prohibited, Solr can either return results where all conditions are met or where one or more conditions are met. The default operator controls this behavior. An operator of `AND` means that all conditions must be fulfilled, while an operator of `OR` means that one or more conditions must be true.

In `schema.xml`, use the `solrQueryParser` element to control what operator is used if an operator is not specified in the query. The default operator setting only applies to the Lucene query parser (not the DisMax query parser, which uses the `mm` parameter to control the equivalent behavior).

SOLRCONFIG.XML

Configuring solrconfig.xml

`solrconfig.xml`, found in the `conf` directory for the Solr Core, comprises of a set of XML statements that set the configuration value for your Solr instance.

AutoCommit

The `<updateHandler>` section affects how updates are done internally. The `<autoCommit>` subelement contains further configuration for controlling how often pending updates will be automatically pushed to the index.

Element	Description
`<maxDocs>`	Number of updates that have occurred since last commit
`<maxTime>`	Number of milliseconds since the oldest uncommitted update

If either of these limits is reached, then Solr automatically performs a commit operation. If the `<autoCommit>` tag is missing, then only explicit commits will update the index.

HTTP RequestDispatcher Settings

The `<requestDispatcher>` section controls how the `RequestDispatcher` implementation responds to HTTP requests.

Element	Description
`<requestParsers>`	Contains attributes for `enableRemoteStreaming` and `multipartUploadLimitInKB`
`<httpCaching>`	Specifies how Solr should generate its HTTP caching-related headers

Internal Caching

The `<query>` section contains settings that affect how Solr will process and respond to queries.

There are three predefined types of caches that you can configure whose settings affect performance:

Element	Description
`<filterCache>`	Used by `SolrIndexSearcher` for filters for unordered sets of all documents that match a query. Solr usese the `filterCache` to cache results of queries that use the `fq` search parameter.
`<queryResultCache>`	Holds the sorted and paginated results of previous searches
`<documentCache>`	Holds Lucene Document objects (the stored fields for each document).

Request Handlers

A Request Handler defines the logic executed for any request. Multiple instances of various request handlers, each with different names and configuration options can be declared. The `qt` url parameter or the path of the url can be used to select the request handler by name.

Most request handlers recognize three main sub-sections in their declaration:

- `default`, which is used when a request does not include a parameter.
- `append`, which is added to the parameter values specified in the request.
- `invariant`, which overrides values specified in the query.

LucidWorks for Solr includes the following indexing handlers:

- `XMLUpdateRequestHandler`: processes XML messages containing data and other index modification instructions.
- `BinaryUpdateRequestHandler`: processes messages from the Solr Java client.
- `CSVRequestHandler`: processes CSV files containing documents
- `DataImportHandler`: processes commands to pull data from remote data sources
- `ExtractingRequestHandler` (aka Solr Cell): uses Apache Tika to process binary files such as Office/PDF and index them

The out-of-the-box searching handler is `SearchHandler`.

Search Components

Instances of `SearchComponent` define discrete units of logic that can be combined together and reused by Request Handlers (in particular `SearchHandler`) that know about them.
The default SearchComponent used by `SearchHandler` is `query`, `facet`, `mlt` (MoreLikeThis), `highlight`, `stats`, `debug`. Additional Search Components are also available with additional configuration.

Response Writers

Response writers generate the formatted response of a search. The `wt` url parameter selects the response writer to use by name. The default response writers are `json`, `php`, `phps`, `python`, `ruby`, `xml`, and `xslt`.

INDEXING

Indexing is the process of adding content to a Solr index, and as necessary, modifying that content or deleting it. By adding content to an index, it becomes searchable by Solr.

Client Libraries

There are a number of client libraries available to access Solr. SolrJ is a Java client included with the Solr 1.4 release which allows clients to add, update and query the Solr index. http://wiki.apache.org/solr/IntegratingSolr provides a list of such libraries.

Indexing Using XML

Solr accepts POSTed XML messages that add/update, commit, delete and delete by query using the http://[hostname]:8983/solr/update url. Multiple documents can be specified in a single `<add>` command.

```
<add>
  <doc>
    <field name="employeeId">05991</field>
    <field name="office">Bridgewater</field>
  </doc>
  [<doc> ... </doc>[<doc> ... </doc>]]
</add>
```

Command	Description
commit	Writes all documents loaded since last commit
optimize	Requests Solr to merge the entire index into a single segment to improve search performance

Delete by id deletes the document with the specified ID (i.e. `uniqueKey`), while delete by query deletes documents that match the specified query:

```
<delete><id>05991</id></delete>
<delete><query>office:Bridgewater</query></delete>
```

Indexing Using CSV

CSV records can be uploaded to Solr by sending the data to the http://[hostname]:8983/solr/update/csv URL.

The CSV handler accepts various parameters, some of which can be overridden on a per field basis using the form:

```
f.fieldname.parameter=value
```

These parameters can be used to specify how data should be parsed, such as specifying the delimiter, quote character and escape characters. You can also handle whitespace, define which lines or field names to skip, map columns to fields, or specify if columns should be split into multiple values.

Indexing Using SolrCell

Using the Solr Cell framework, Solr uses Tika to automatically determine the type of a document and extract fields from it. These fields are then indexed directly, or mapped to other fields in your schema.

The URL for this handler is http://[hostname]:8983/solr/update/extract.

The Extraction Request Handler accepts various parameters that can be used to specify how data should be mapped to fields in the schema, including specific XPaths of content to be extracted, how content should be mapped to fields, whether attributes should be extracted, and in which format to extract content. You can also specify a dynamic field prefix to use when extracting content that has no corresponding field.

Indexing Using Data Import Handler

The Data Import Handler (DIH) can pull data from relational databases (through JDBC), RSS feeds, emails repositories, and structure XML using XPath to generate fields.

The Data Import Handler is registered in `solrconfig.xml`, with a pointer to its `data-config.xml` file which has the following structure:

```
<dataConfig>
  <dataSource/>
  <document>
   <entity>
     <field column="" name=""/>
     <field column="" name=""/>
    </entity>
  </document>
</dataConfig>
```

The Data Import Handler is accessed using the http://[hostname]:8983/solr/dataimport URL but it also includes a browser-based console which allows you to experiment with `data-config.xml` changes and demonstrates all of the commands and options to help with development. You can access the console at this address: http://[hostname]:port/solr/admin/dataimport.jsp

SEARCHING

Data can be queried using either the http://[hostname]:8983/solr/select?qt=name URL, or by using the http://[hostname]:8983/solr/name syntax for SearchHandler instances with names that begin with a "/".

`SearchHandler` processes requests by delegating to its Search Components which interpret the various request parameters. The `QueryComponent` delegates to a query parser, which determines which documents the user is interested in. Different query parsers support different syntax.

Query Parsing

Input to a query parser can include:

- Search strings—that is, terms to search for in the index.
- Parameters for fine-tuning the query by increasing the importance of particular strings or fields, by applying Boolean logic among the search terms, or by excluding content from the search results.
- Parameters for controlling the presentation of the query response, such as specifying the order in which results are to be presented or limiting the response to particular fields of the search application's schema.

Search parameters may also specify a filter query. As part of a search response, a filter query runs a query against the entire index and caches the results. Because Solr allocates a separate cache for filter queries, the strategic use of filter queries can improve search performance.

Common Query Parameters

The table below summarizes Solr's common query parameters:

Parameter	Description
`defType`	The query parser to be used to process the query
`sort`	Sort results in ascending or descending order based on the documents score or another characteristic
`start`	An offset (0 by default) to the results that Solr should begin displaying
`rows`	Indicates how many rows of results are displayed at a time (10 by default)
`fq`	Applies a filter query to the search results
`fl`	Limits the query's results to a listed set of fields
`debugQuery`	Causes Solr to include additional debugging information in the response, including score explain information for each document returned
`explainOther`	Allows client to specify a Lucene query to identify a set of documents not already included in the response, returning explain information for each of those documents
`wt`	Specified the Response Writer to be used to format the query response

Lucene Query Parser

The standard query parser syntax allows users to specify queries containing complex expressions, such as: .
http://[hostname]:8983/solr/select?q=id:SP2514N+price:[*+TO+10].

The standard query parser supports the parameters described in the following table:

Parameter	Description
`q`	Query string using the Lucene Query syntax
`q.op`	Specified the default operator for the query expression, overriding that in `schema.xml`. May be `AND` or `OR`
`df`	Default field, overriding what is defined in `schema.xml`

DisMax Query Parser

The DisMax query parser is designed to provide an experience similar to that of popular search engines such as Google, which rarely display syntax errors to users.

Instead of allowing complex expressions in the query string, additional parameters can be used to specify how the query string should be used to find matching documents.

Parameter	Description
`q`	Defines the raw user input strings for the query
`q.alt`	Calls the standard query parser and defined query input strings, when q is not used
`qf`	Query Fields: the fields in the index on which to perform the query
`mm`	Minimum "Should" Match: a minimum number of clauses in the query that must match a document. This can be specified as a complex expression.

`pf`	Phrase Fields: Fields that give a score boost when all terms of the q parameter appear in close proximity
`ps`	Phrase Slop: the number of positions all terms can be apart in order to match the pf boost
`tie`	Tie Breaker: a float value (less than 1) used as a multiplier with more then one of the qf fields containing a term from the query string. The smaller the value, the less influence multiple matching fields have
`bq`	Boost Query: a raw Lucene query that will be added to the users query to influence the score
`bf`	Boost Function: like bq, but directly supports the Solr function query syntax

ADVANCED SEARCH FEATURES

Faceting makes it easy for users to drill down on search results on sites such as movie sites and product review sites, where there are many categories and many items within a category.

There are three types of faceting, all of which use indexed terms:

- **Field Faceting:** treats each indexed term as a facet constraint.
- **Query Faceting:** allows the client to specify an arbitrary query and uses that as a facet constraint.
- **Date Range Faceting:** creates date range queries on the fly.

Solr provides a collection of **highlighting utilities** which can be called by various Request Handlers to include highlighted matches in field values. Popular search engines such as Google and Yahoo! return snippets in their search results: 3-4 lines of text offering a description of a search result.

When an index becomes too large to fit on a single system, or when a query takes too long to execute, the index can be split into multiple shards on different Solr servers, for **Distributed Search**. Solr can query and merge results across shards. It's up to you to get all your documents indexed on each shard of your server farm. Solr does not include out-of-the-box support for distributed indexing, but your method can be as simple as a round robin technique. Just index each document to the next server in the circle.

Clustering groups search results by similarities discovered when a search is executed, rather than when content is indexed. The results of clustering often lack the neat hierarchical organization found in faceted search results, but clustering can be useful nonetheless. It can reveal unexpected commonalities among search results, and it can help users rule out content that isn't pertinent to what they're really searching for.

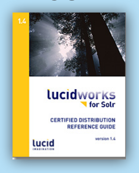
The primary purpose of the **Replication Handler** is to replicate an index to multiple slave servers which can then use load-balancing for horizontal scaling. The Replication Handler can also be used to make a back-up copy of a server's index, even without any slave servers in operation.

MoreLikeThis is a component that can be used with the `SearchHandler` to return documents similar to each of the documents matching a query. The `MoreLikeThis` Request Handler can be used instead of the `SearchHandler` to find documents similar to an individual document, utilizing faceting, pagination and filtering on the related documents.

ABOUT THE AUTHOR

Chris Hostetter is Senior Staff Engineer at Lucid Imagination, a member of the Apache Software Foundation, and serves as a committer for the Apache Lucene/Solr Projects. Prior to joining Lucid Imagination in 2010 to work full time on Solr development, he spent 11 years as a Principal Software Engineer for CNET Networks thinking about searching "structured data" that was never as structured as it should have been.

RECOMMENDED BOOK



Designed to provide complete, comprehensive documentation, the Reference Guide is intended to be more encyclopedic and less of a cookbook. It is structured to address a broad spectrum of needs, ranging from new developers getting started to well experienced developers extending their application or troubleshooting. It will be of use at any point in the application lifecycle, for whenever you need deep, authoritative information about Solr.

Download Now
http://bit.ly/solrguide

DZone, Inc.
140 Preston Executive Dr.
Suite 100
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-02-8
ISBN-10: 1-936502-02-X
50795
9 781936 502028
$7.95

Version 2.0