

DMLDoc: Installation Guide

I₁

dmldoc_std_01

Software Release: DMLDoc Standard Edition 01

Documentation Release: 01

S e t I n f o r m a t i o n


<http://www.set-i.com/products/>

Documentation Catalogue

DMLDoc: Installation Guide - this document.

DMLDoc: Web Designer's Guide - Describes the application of the DML language to the task of creating templates for dynamically-generated HTML pages. Essential reading for anyone creating Web pages for DMLDoc projects. Assumes some knowledge of HTML, and no knowledge of Java or other programming languages.

DMLDoc: Programmers Guide - Describes the use of DMLDoc in Java server-side programming. Covers the architecture of the runtime environment and the structure of DMLDoc applications. The guide steps through the construction of DMLDoc applications, based on example code which can be got from <http://www.set-i.com/products/dmldoc/examples/>. Assumes some knowledge of Java programming.

DMLDoc JavaDocs - A complete doclet of JavaDocs for all the packages and classes for DMLDoc. Essential reading for DMLDoc server-side programmers. Available for download or online at: <http://www.set-i.com/products/dmldoc/1.1/docs/api/>.

DMLDoc: Technology Guide - Discusses how the DMLDoc packages can be extended in order to change or extend the DMLDoc data structures and behaviour. The guide describes the structures used by DMLDoc in detail, and gives pointers to the methods and fields detailed in the DMLDoc JavaDoc documentation. Assumes advanced Java programming knowledge.

DMLDoc White Papers & Roadmap - A number of documents covering the philosophy behind DMLDoc and future plans for products.

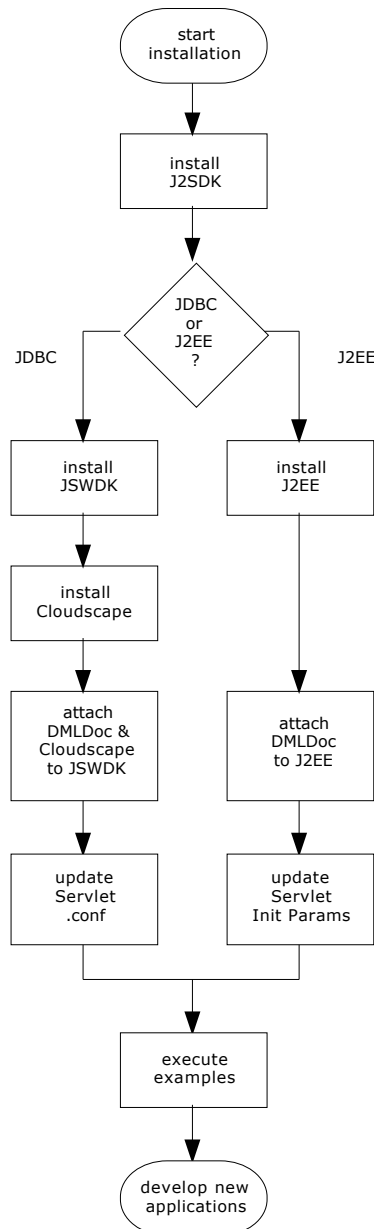
All documents in this catalogue are Copyright © 2000 Set Information Limited, 7 Wyndham Street, Brighton, UK. All Rights reserved. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, inc. in the U.S. and other countries. Other trademarks are the property of their respective owners.

Contents

1	Runtime Installation	4
1.1	... Installing the J2SDK	5
1.2	... Alternate Data Sources	6
1.3	... JSWDK	7
1.3.1 Cloudscape	8
1.3.2 JSWDK Classpath	9
1.3.4conf Configuration	10
1.4	... J2EE	11
1.4.1 J2EE Classpath	12
1.4.2 J2EE Servlet Initialisation	13
1.5	... Executing Examples	14
2	IDE Installation	15
2.1	... Alternate IDEs	16
2.2	... Importing DMLDoc Classes	17
2.3	... Including DMLDoc Classes	18
2.4	... Editing Source Code	19
2.5	... Compiling Source Code	20
2.6	... Exporting Object Code	21
2.7	... Delivering the Application	22
2.8	... Executing Applications	23
A	Software Sources	24
B	Contact Details	25

1

Runtime Installation



This section deals with the process of setting up a runtime environment that can make use of DMLDoc. The flowchart on the left shows the whole process - subsequent subsections step through each stage, one at a time.

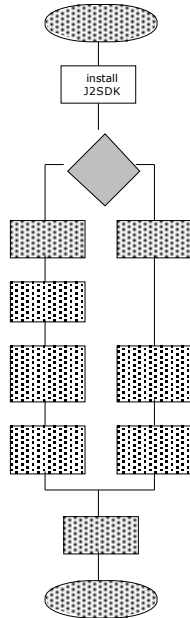
The example applications supplied with DMLDoc give a context for the setup process. The steps described here should take you from an 'empty' machine through to running the examples. You might want to examine the read.me files that accompany the examples in order to check the fine detail.

The flowchart shows the JDBC and J2EE routes as alternatives. In practice, you might want to try both, if only out of curiosity. Neither route is particularly hard work, or particularly complicated.

At some points, the flow chart uses the names of specific products, such as 'J2EE' and 'Cloudscape'. These are products that allow free download (sometimes of evaluation copies) and are known to work with DMLDoc and the DMLDoc examples. In such cases, these can be swapped for other products - we have been careful to avoid any non-standard features of database drivers, etc.

Once you have completed the stages in this section, you can develop new DMLDoc applications. Section 2 'IDE Installation' explains how to incorporate the DMLDoc packages into different Java IDEs, or use DMLDoc with the javac compiler.

1.1

Runtime Installation: **Installing the J2SDK**

Note: If you are familiar with Java server-side programming, then you should certainly have a suitable Java VM installed on your machine, and you can skip this stage.

DMLDoc requires the Java 2 Virtual Machine (VM) in order to run. The Java 2 VM can be downloaded from:

<http://java.sun.com/j2se/>

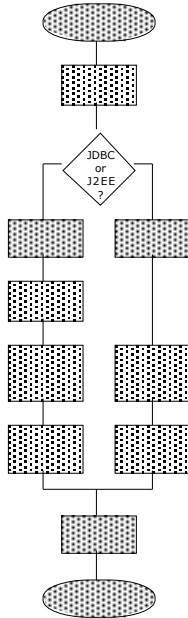
The Java VM is supplied either as a 'Runtime' (JRE) or as a 'Software Development Kit' (SDK). We strongly recommend that you get the SDK - the JRE will allow you to run DMLDoc applications, but will not allow you to recompile the Java source code supplied with the examples - you will most likely want to recompile these sources.

Note the naming confusion with Java - for 1.0.x and 1.1.x versions of Java, the development tools were referred to as the Java Development Kit (JDK). With 1.2.x, Java became known as Java 2, and the Java Development Kit became known as the Java 2 Software Development Kit (J2SDK).

DMLDoc was developed using a number of different versions of the 1.2.2 J2SDK. The current version of the J2SDK is 1.3. DMLDoc should work fine with all the 1.3 J2SDK versions.

The J2SDK is available for Solaris, Linux, and Windows. Running the J2SDK on other versions of UNIX may be more-or-less straightforward, consult your UNIX supplier.

1.2

Runtime Installation: **Alternate Data Sources**

The examples provided with the DMLDoc Standard Edition, Version 1 have three different data source requirements:

- `demo_jswdk` - Requires no data source. The purpose of this example is simply to test that the Java Servlet environment is operating, and that the DMLDoc installation procedures have been successful.
- `demo_jdbc` - Requires a Java Database Connectivity (JDBC) database connection. The example will work with any JDBC database driver that conforms to the Java 2 `java.sql` API.
- `demo_j2ee` - Requires an Enterprise Java Bean (EJB) server. The Java 2 Enterprise Edition (J2EE) components, available from java.sun.com provide all of the required functionality.

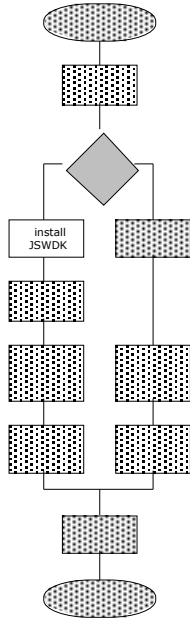
All the examples require a Java Servlet container:

In the case of `demo_jswdk` and `demo_jdbc`, this can be provided by the Java Servlet Web Development Kit (JSWDK) Web server and servlet container. Other Web server / servlet container combinations - such as the Apache Web server and JServ servlet container - work equally well.

In the case of `demo_j2ee`, the J2EE components include a suitable servlet container, so the JSWDK is not required.

The following subsections cover the installation of all of these runtime components.

1.3

Runtime Installation: **JSWDK**

Note: This stage is not required if you are only going to use the J2EE components.

All DMLDoc applications, including all the supplied examples, require a Java servlet container in order to run. There are numerous servlet containers on the market. A full list of such products is available at:

<http://java.sun.com/products/servlet/industry.html>

If you have not used servlets before, we recommend that you install the java.sun.com Java Servlet Web Development Kit (JSWDK). This is available for free download from:

<http://java.sun.com/products/jsp/download.html>

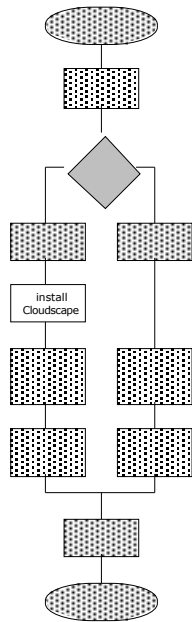
The JSWDK includes the Tomcat Web server. One of the best features of the JSWDK is that you do not need to get involved in the details of Web server to servlet container protocols - the whole thing works right out of the box (or, in this case, out of the download).

Before proceeding with the DMLDoc installation, you should verify that any servlet examples provided with the servlet container run properly.

The JSWDK requires the J2SDK, and should run properly wherever the J2SDK runs. However, take care to ensure that any other TCP/IP port listeners (such as other Web servers) do not conflict with the port used by the JSWDK Web server, normally port 8080.

Note that you must also add and amend certain files in the JSWDK directory - check the read.me for each example.

1.3.1 Runtime Installation: **Cloudscape**



Note: This stage is not required if you are only going to use the J2EE components.

The `demo_jdbc` example requires a JDBC data source. You may already have a suitable JDBC data source installed on your machine - if so, you can ignore this stage.

If you have no JDBC data source currently installed on your machine, we recommend the Cloudscape database. This is because the Cloudscape database is included among the J2EE components. Thus, if you go on to install and run the `demo_j2ee` example using the J2EE, potential database hazards, misunderstandings and incompatibilities will be avoided.

The Cloudscape database is available for free evaluation download from:

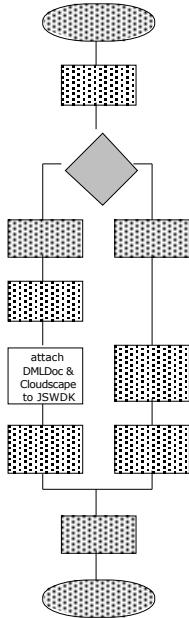
<http://www.cloudscape.com/>

The Cloudscape database driver is written in Java, and will run wherever a suitable Java runtime is available. In our experience, Cloudscape will run with any Java 1.1.8 or 1.2.2 runtime.

Cloudscape is supplied with a graphical database administration tool, called Cloudview. This tool is not required by the DMLDoc examples, but it does provide some useful feedback about the way in which the DMLDoc JDBC and J2EE examples operate. Cloudview requires the Java SWING classes and hence is effectively limited to the J2SDK.

Configuring Cloudview is not a simple task, but the Cloudview documentation is helpful in this area.

1.3.2

Runtime Installation: **JSWDK Classpath**

Note: This stage is not required if you are only going to use the J2EE components.

The servlet container must be made aware of the Java classes that are required by the applications that are hosted by the container.

For the DMLDoc examples - and any other DMLDoc applications that make use of JDBC connections - the servlet container must be made aware of the database driver, the DMLDoc package and the DMLDoc license.

References to these classes are made on the classpath specified for the container when it is run. For the JSWDK, this is done in the script called `startserver`. For Windows environments, this is a `.BAT` file, for all varieties of UNIX, this is a shell script.

The Java Archive (JAR) files that should be included on the CLASSPATH for JSWDK are:

```

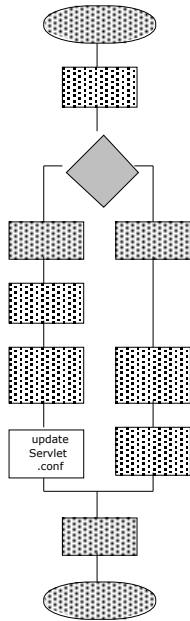
${CLOUDSCAPE_INSTALL}/lib/license.jar
${CLOUDSCAPE_INSTALL}/lib/cloudscape.jar
${CLOUDSCAPE_INSTALL}/lib/tools.jar

${DMLDOC_INSTALL}/dmldoc_std_011.jar
${DMLDOC_INSTALL}/dmldoc_std_lic.jar

```

Where `${CLOUDSCAPE_INSTALL}` is the complete path specification of the directory created by the Cloudscape installation, and `${DMLDOC_INSTALL}` identifies the directory holding the DMLDoc JAR files. A `startserver.sh` and `startserver.bat` file is included with each example.

1.3.3 Runtime Installation: **.conf Configuration**



Note: This stage is not required if you are only going to use the J2EE components.

The `demo_jswdk` and `demo_jdbc` examples are supplied with configuration files. These files allow you to update any environment-specific settings for the example servlets. Thus, it is possible to move the servlets from one host to another without recompiling the servlets' source code. (A similar effect can be achieved using servlet Initialisation Parameters, but in many cases the configuration file is a superior approach.)

The examples make use of configuration files named for the servlet classes. Thus, the `demo_jswdk` example has two configuration files:

```
com.set_i.demo.jswdk.ContactServlet.conf
com.set_i.demo.jswdk.UploadServlet.conf
```

The `demo_jdbc` example has one configuration file:

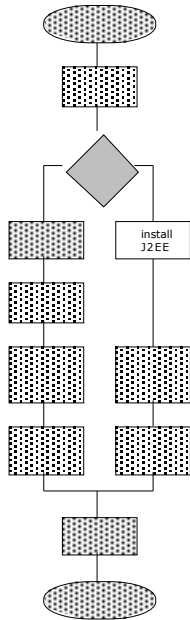
```
com.set_i.demo.jdbc.ContactServlet.conf
```

In order that the configuration files can be found at runtime, they too must be accessible to the servlet container. The easiest way to do this is to edit the configuration files held in the classes directory of each example. Thus, you should edit the files:

```
classes/com/set_i/demo/jswdk/ContactServlet.conf
classes/com/set_i/demo/jswdk/UploadServlet.conf
classes/com/set_i/demo/jdbc/ContactServlet.conf
```

to reflect your file system and URL.

1.4

Runtime Installation: **J2EE**

Note: This stage is not required if you are only going to use JDBC data sources.

The J2EE provides an environment in which you can build transactional, server-side applications with minimum involvement of connectivity and storage details.

If you are not completely familiar with the J2EE principles, we urge you to read a paper entitled 'Simplified Guide to the Java 2 Platform, Enterprise Edition', available from:

<http://java.sun.com/j2ee/overview.html>

The J2EE implementation available from java.sun.com provides a 'reference standard' platform that can be used to support applications - such as that given in `demo_j2ee` - which make use of J2EE services. It is by no means the easiest to use or the most efficient implementation of these services. Other IT vendors providing products in this area can be found at:

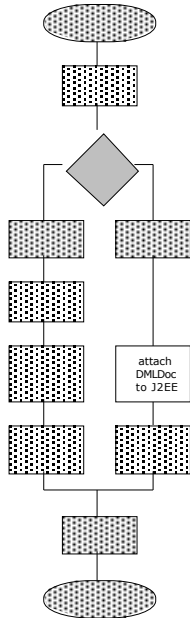
<http://java.sun.com/j2ee/industry.html>

The java.sun.com J2EE can be downloaded free of charge at:

<http://java.sun.com/j2ee/>

As with the JSWDK, J2EE requires the J2SDK, and should run properly wherever the J2SDK runs. However, take care to ensure that any other TCP/IP port listeners (such as other Web servers) do not conflict with the port used by the J2EE Web server, normally port 8000.

1.4.1

Runtime Installation: **J2EE Classpath**

Note: This stage is not required if you are only going to use JDBC data sources.

As with the Java Servlet Web Development Kit, the J2EE server and deployment tool must be made aware of the DMLDoc classes and license.

In the case of J2EE, this is very easy to do: simply add the relevant DMLDoc JAR files on the J2EE classpath, as specified in the J2EE `userconfig.sh` or `userconfig.bat` file. The JAR files are:

```
${DMLDOC_INSTALL}/dmldoc_std_011.jar
${DMLDOC_INSTALL}/dmldoc_std_lic.jar
```

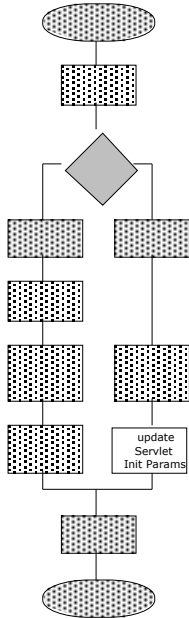
Where `${DMLDOC_INSTALL}` is the complete path specification of the directory containing the DMLDoc files. Sample `userconfig.sh` or `userconfig.bat` files are included with the `demo_j2ee` example.

Unlike the JSWDK, the J2EE classpath does not need the Cloudscape components to be included. This is because, for J2EE, the database driver runs in a separate Java process.

The J2EE download from java.sun.com includes a Cloudscape database. Instructions for running the database, the J2EE server and the J2EE deployment tool are included with the download.

You should make yourself familiar with the simpler J2EE example applications - also included with the J2EE download - before proceeding.

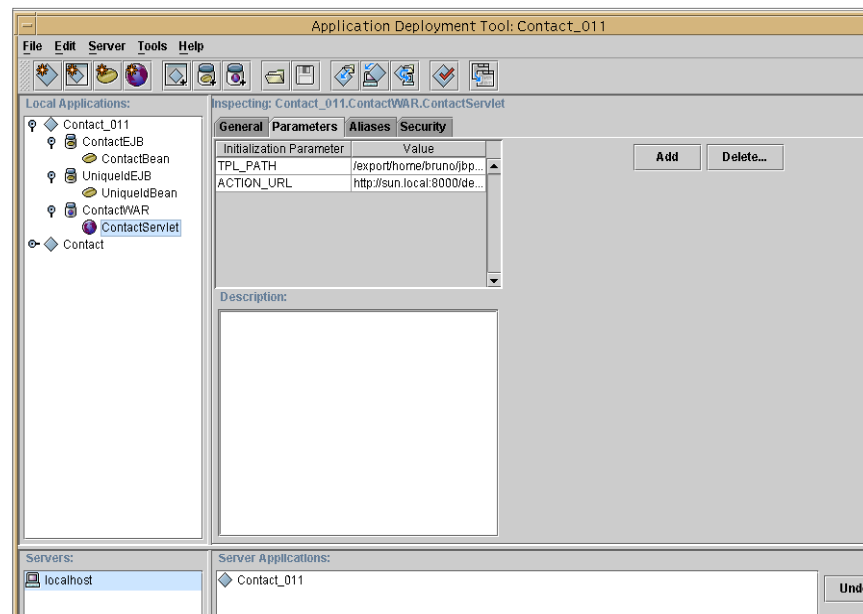
1.4.2 Runtime Installation: **J2EE Servlet Initialisation**



Note: This stage is not required if you are only going to use JDBC data sources.

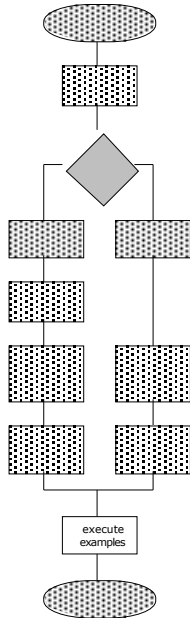
With the JSWDK examples, we used `.conf` files to hold environment-specific values for each servlet. However, using `.conf` files is less appropriate where J2EE provides the servlet container. This is because specifying a `.conf` file as part of the application's Web component is more tricky, and in part because the application description provided by the deployment tool server the same purpose as the `.conf` file.

The `demo_j2ee` example therefore uses servlet Initialisation Parameters. To update these, use the J2EE deploytool to open the `Contact_011.ear` file provided with the `demo_j2ee` example.



Select the `ContactServlet` Parameters Tag, update the two parameters with suitable values, then deploy the application.

1.5

Runtime Installation: **Executing Examples**

For all of the DMLDoc examples, the only means of accessing the applications is via a Web browser. Although it is possible to build stand-alone Java clients for J2EE applications, in practice this is not necessary.

The interaction between browser and server-side application is largely independent of the server-side container. Thus, JSWDK and J2EE examples operate in the same way from the user's perspective - the only real difference is the form of the URL.

If you have installed the JSWDK examples according to their `read.me` files, the application can be accessed with, for example:

```
http://localhost:8080/demo/servlet/contactDb
```

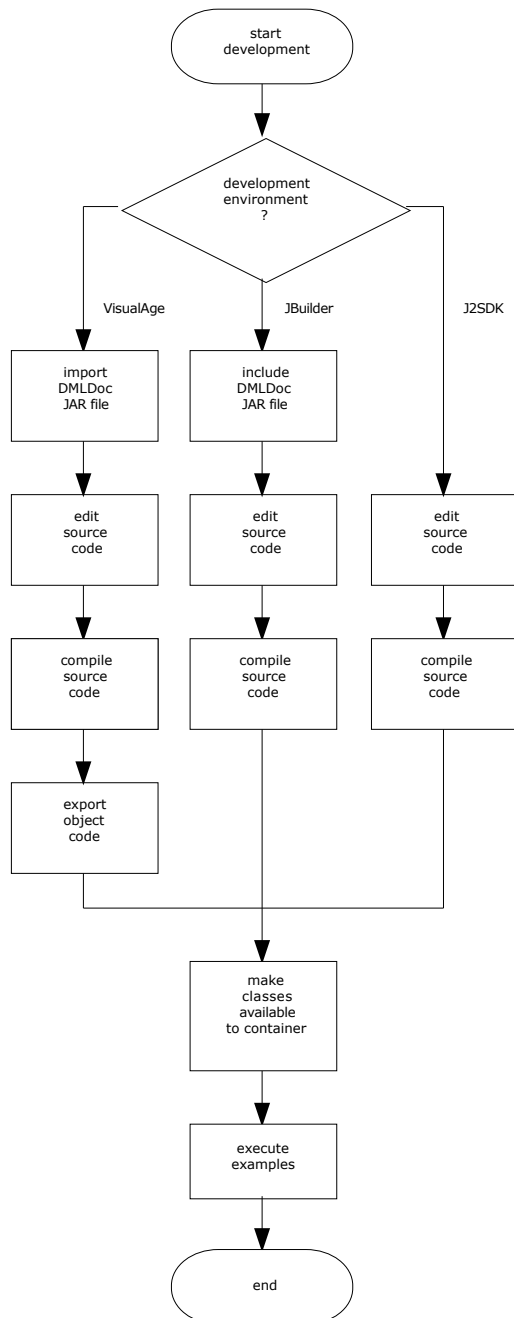
For the above URL, no parameters have been specified, so the servlet will respond with an 'unknown command' message and will list all the possible parameters and values. For this JDBC example, the first command issued should be `cmd=create_db` to create the necessary database tables.

In the case of the J2EE examples, the database tables used to provide persistence for the Enterprise Java Beans (EJBs) are created automatically on deployment. Once deployed, the J2EE application can be accessed with, for example:

```
http://localhost:8000/demo/contact
```

Be sure to shut down the server - JSWDK or J2EE - in the correct way. Failure to do this is likely to create database corruption, and cause unexpected results.

IDE Installation



DMLDoc applications make reference to DMLDoc packages using the standard Java technique - at the start of each Java source file the appropriate package is specified in an `include` statement.

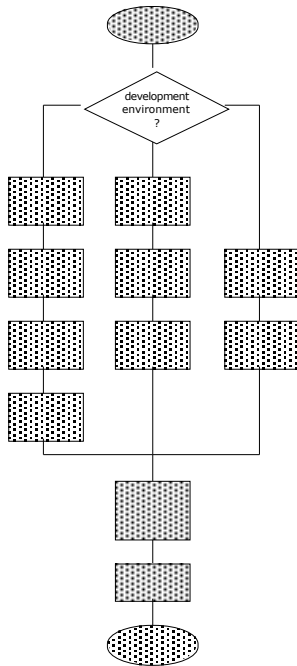
Strictly speaking, it is not necessary to introduce the DMLDoc packages in your Integrated Development Environment (IDE) until you are ready to compile one or more classes. However, some IDEs - such as Borland JBuilder - provide statement completion or automatic method and field listings at the cursor as the source text is edited.

When a package such as `com.set_i.dmldoc` or `com.set_i.http` is introduced to the IDE, it should automatically be added to the `javac` compiler classpath. If the J2SDK is used without an accompanying integrated development environment, then the DMLDoc packages must be named on the compiler's classpath.

The DMLDoc packages should also be made available to the runtime system - the methods for doing this are described in Section 1.

Note that the compiler or IDE only needs access to the `dmldoc_std_011.jar` file, whereas the runtime system also needs access to the `dmldoc_std_lic.jar` license file.

2.1

IDE Installation: **Alternate IDEs**

IDEs for Java vary in the way in which the DMLDoc packages must be introduced to the IDE before such an include can be effective. The important factor here is whether the IDE operates using a managed repository of Java classes, or whether it is able to obtain classes from the local file system.

The Java Integrated Development Environment may use a shared repository (such as IBM TeamConnection), it may use a personal repository (the stand-alone version of IBM VisualAge takes this approach), or it may use a local or networked file system.

The last approach is used by Borland JBuilder and Forté for Java. For this arrangement, the `dmldoc_std_011.jar` file remains in the file system and does not need to be unpacked.

The J2SDK javac compiler uses the `dmldoc_std_011.jar` file in the same way. The file is introduced to the compiler by a command line switch.

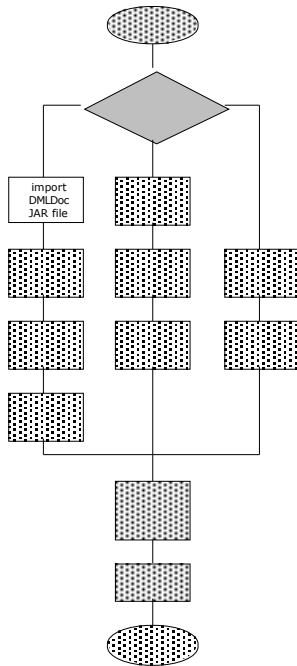
Products such as Forté for Java also allow integration with the Concurrent Versions System (CVS) control system. This can be used to manage DMLDoc versions as well as application source code.

Whatever approach is taken, some care should be exercised in managing the DMLDoc packages - new versions will become available periodically, and these new versions may not share the same API. It is therefore not a good idea to switch versions of DMLDoc mid-project, unless the switch solves a pressing problem.

2.2

IDE Installation: **Importing DMLDoc Classes**

NOTE: This stage only applies to IBM VisualAge and other repository-based IDEs.



Using IBM VisualAge, it is possible to import the complete collection of DMLDoc packages in one operation - simply perform an Import on the `dmldoc_std_011.jar` file.

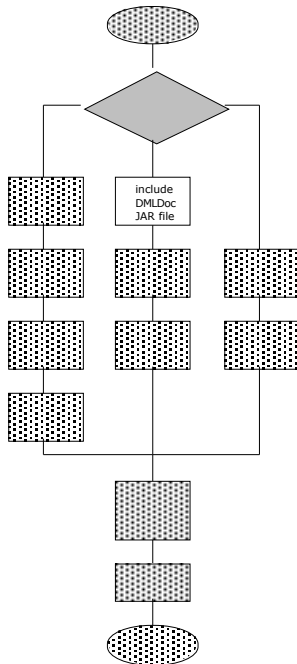
Once this is done, all of the DMLDoc packages and classes will be visible within the VisualAge class browser. You can now Import any or all of the DMLDoc example source code, and proceed with development work.

If you want to upgrade to a new version of DMLDoc, you must first delete all of the existing DMLDoc classes within your VisualAge repository. In order to do this, you must be sure which these classes are! Check the DMLDoc JavaDocs for the old version if you are not sure.

2.3

IDE Installation: **Including DMLDoc Classes**

NOTE: This stage only applies to Borland JBuilder, Forté for Java and other file-based IDEs.



If you are using Borland JBuilder, or any other file-based Java IDE such as Symantec Visual Café, you can simply include a reference to the `dmldoc_std_011.jar` file.

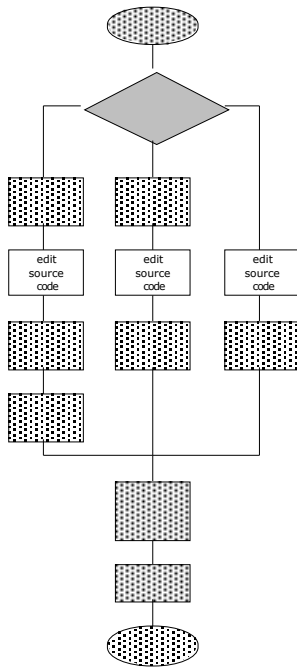
In the case of JBuilder, the `dmldoc_std_011.jar` file can be included as a Required Library in either the Project Properties of one or more specific projects, or in the Default Project Properties, making its contents available to all projects. Given that different versions of DMLDoc may be used over time, the former approach is the safest.

In the case of Forté for Java, you can mount the `dmldoc_std_011.jar` file directly, as part of the file system known to the IDE. Do this by right-clicking on the File systems node in the Explorer and choosing Add JAR... from the contextual menu.

To change versions of DMLDoc using JBuilder, select the DMLDoc Required Library in the Project Properties dialogue and click on Remove. Then add the new DMLDoc Jar file. It is a good idea to include the DMLDoc version number in the Library entry name.

To change versions of DMLDoc using Forté for Java, right-click on the Jar file and choose Remove From File systems. Then add the new version.

2.4

IDE Installation: **Editing Source Code**

There are no special source code editing requirements for DMLDoc. With most Java IDEs, you can treat references to DMLDoc classes, methods and public fields in the same way as references to elements in your own application code.

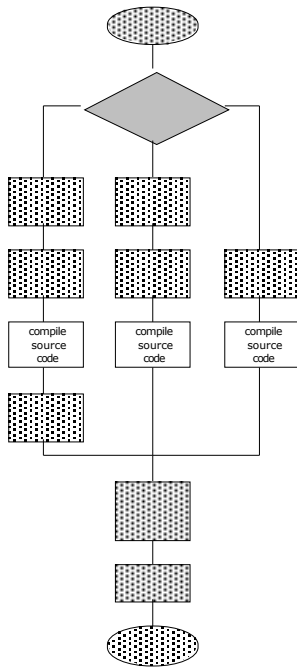
Some IDEs - such as JBuilder and Forté for Java - include dynamic code completion features, and DMLDoc entities are included into these systems.

For JBuilder, depending on how your editor is configured, code completion menus appear when you type the '.' after a variable of a recognised class, or other recognisable identifier. For Forté for Java, the same effect is achieved by holding down the [CTRL]+[SPACE] keys, or pausing after typing the '.' character.

Some IDEs allow class definitions to be associated with JavaDoc pages. Alternatively, the IDE may allow JavaDoc 'doclets' to be imported into the IDE's help system. The DMLDoc JavaDocs may be used in this way.

Developers are welcome to use the source code provided in the DMLDoc examples in any way - there are no restrictions on the use of this source code.

2.5

IDE Installation: **Compiling Source Code**

Assuming that the DMLDoc packages have been correctly imported into the IDE - and that a J2SDK 1.2.2 or greater compiler is being used - compiling DMLDoc applications should present no special problems.

In the case of the J2SDK javac compiler, the DMLDoc packages can be introduced thus:

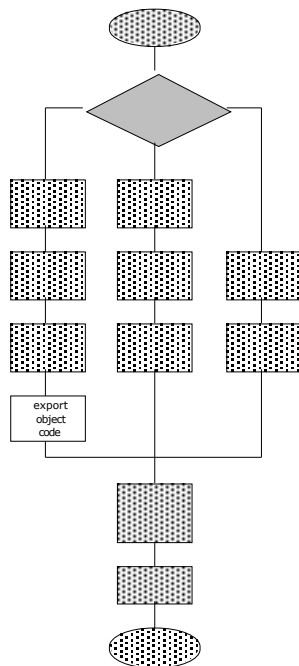
```
javac -classpath ../../dmldoc_std_011.jar:../../servlet.jar \
-d ../classes com/set_i/demo/jswdk/*.java
```

The example assumes that you are in the directory `src`, and that you want to update the `.class` file in the subdirectories of `classes`. The archive referred to as `../../servlet.jar` is one of the JAR files supplied with the JSWDK.

2.6

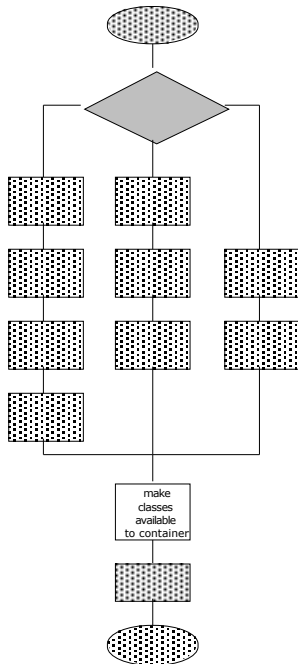
IDE Installation: **Exporting Object Code**

NOTE: This stage only applies to IBM VisualAge and other repository-based IDEs.



When using a repository-based IDE, it is necessary to export the application code once the development cycle is complete. The application code should be exported as a nested directory of Java `.class` files.

2.7

IDE Installation: **Delivering the Application**

In Sections 1.3.2 (JSWDK Classpath) and 1.4.1 (J2EE Classpath) we looked at how to make the DMLDoc packages available to two different runtime systems. In this section, we look at how to make compiled DMLDoc applications available to the same runtime systems.

The examples `demo_jswdk` and `demo_jdbc` both contain a directory called `demo`. One of these two directories should be placed in the JSWDK directory installed on your machine. The JSWDK `webserver.xml` file should then be edited so that its `webserver` tag should read:

```
<WebServer id="webServer">
  <Service id="service0">
    <WebApplication id="examples"
      mapping="/examples" docBase="examples"/>
    <WebApplication id="demo"
      mapping="/demo" docBase="demo"/>
  </Service>
</WebServer>
```

Once this is done, the JSWDK Web server will now direct URLs containing `demo/servlet/...` to the correct place.

Next, you must put the compiled application classes into the `demo/WEB-INF/servlets` directory - you can either move the `com` directory for the compiled application classes into the `demo/WEB-INF/servlets` directory, or you can create a link from the `demo/WEB-INF/servlets` directory to that `com` directory.

The file at `demo/WEB-INF/servlets.properties` defines the way in which URLs are mapped onto Java classes - this file should not need to be edited.

The example `demo_j2ee` contains a file called `Contact_011.ear`. This is a J2EE application description file, and it can be read by the J2EE deploytool.

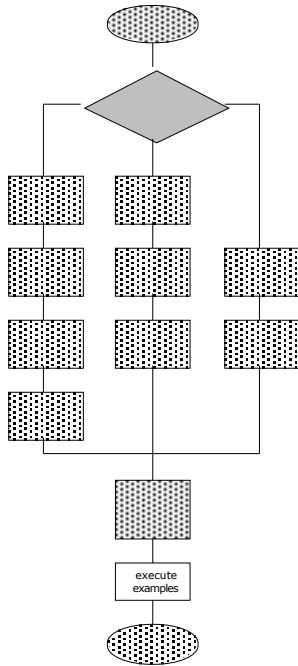
Using the J2EE deploytool, open the `Contact_011.ear` file. This will provide you with the complete application assembly for the J2EE DMLDoc example. Before deploying the example application, you must update the Initialisation Parameters for the example's servlet - see section 1.4.2 (J2EE Servlet Initialisation) for instruction on how to do this.

You can choose to assemble the application yourself using the J2EE deploytool, or use the deploytool to mix and match components from the `Contact_011.ear` file with your own components. The following is a description of the assembly:

- `ContactWAR` - Contains only one servlet, namely `com.set_i.demo.j2ee.ContactServlet.class`. The alias for the WAR component should be `contact`.
- `ContactEJB` - Contains the three standard parts of an EJB, namely: `com.set_i.demo.j2ee.ContactEJB.class` - the Enterprise Bean class; `com.set_i.demo.j2ee.ContactHome.class` - the Home Interface; `com.set_i.demo.j2ee.Contact.class` - the Remote Interface. The Contact EJB should be given the alias `ContactService`.
- `UniqueIdEJB` - Again, this contains three standard parts, namely: `com.set_i.demo.j2ee.UniqueIdEJB.class` - the Enterprise Bean class; `com.set_i.demo.j2ee.UniqueIdHome.class` - the Home Interface; `com.set_i.demo.j2ee.UniqueId.class` - the Remote Interface. The UniqueId EJB should be given the alias `UniqueIdService`.

The application as a whole should be given the Web alias `demo`.

2.8

IDE Installation: **Executing Applications**

The process of executing examples is covered in Section 1.5 (Executing Examples).

The first time a DMLDoc servlet is executed, the DMLDoc version and license information is written to the `System.err` channel of the servlet container - `System.err` may be directed to a console or to a error file, depending on which servlet container is used, and how it is configured.

The examples may also write other information to the channel in order to report on servlet initialisation and shut down.

All of the examples produce an automatic HTML report if obligatory parameters are missing, or if the parameter values are not meaningful to the servlet. This behaviour can be changed, as can the report format. Refer to the [DMLDoc Programmer's Guide](#) for more information on this behaviour.

A **Software Sources**

J2SDK - DMLDoc has been tested with a number of different versions of the J2SDK 1.2. The most recent version is 1.3. A suitable Java Development Kit (JDK, now known as the Java 2 Platform, Standard Edition or J2SDK) can be downloaded from:

<http://java.sun.com/j2se/>

JSWDK - The Java Servlet Web Development Kit (JSWDK) provides a suitable servlet container. (DMLDoc applications should work with any standard servlet container.) The JSWDK 1.0.1 is available from:

<http://java.sun.com/products/jsp/download.html>

J2EE - The Java 2 Enterprise Edition (J2EE) provides a Web server, servlet container and EJB container, and database. The J2EE was used to develop the DMLDoc EJB features. This software can be downloaded from:

<http://java.sun.com/j2ee/>

Cloudscape - The J2EE database is the Cloudscape 3.0 database, owned by Informix. The Cloudscape 3.0 database was used to develop the JDBC-related features of DMLDoc. The latest version (Cloudscape 3.5) can be downloaded from:

<http://www.cloudscape.com/>

Apache and **JServ** - This is a high-performance, open-source Web server and servlet container, available from:

<http://www.apache.org/> and: <http://java.apache.org/>

B **Contact Details**

DMLDoc is developed and distributed by:

Set Information Limited

7 Wyndham Street
Brighton
East Sussex BN2 1AF
United Kingdom

Tel: +44 1273 680 806

Fax: +44 1273 699 224

Email: contact@set-i.com

Web: <http://www.set-i.com/products/>

Set Information is registered in England, No. 3851469.

No formal support arrangements are available for the Standard Edition of DMLDoc. The software, like the documentation, is supplied *as is*, with no warranty whatsoever as to the suitability of the product for any task. Set Information Limited cannot accept any liability or consequential liability for the operation of DMLDoc, or its possible non-operation.

Legalese aside, *Set Information is very eager to hear from users of DMLDoc* - we welcome bug reports and suggestions for new features. We will do our best to help with installation or programming problems. A FAQ for DMLDoc designers and programmers will be available shortly.

If you are using DMLDoc for a specific project, please let us know - your experiences will most likely be useful to future developers.