This presentation provides an in-depth technical overview of the loading utilities for WebSphere Commerce version 6

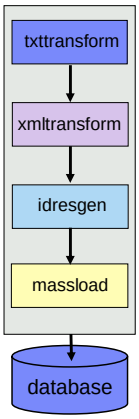
Agenda

- Introduction to the WebSphere Commerce Loading Utilities
- Technical deep-dive of each utility
  - txttransform
  - xmltransform
  - idresgen
  - massload
- An example is incorporated into the explanation of each utility

The following points are discussed during the course of this presentation.

You will see an overview of what the WebSphere Commerce loading utilities are, followed by more technical details of each of the loading utilities. The loading utilities in WebSphere Commerce version 6 consists of the txttransform utility, the xmltransform utility, the idresgen utility and the massload utility.

Introduction to the loading utilities



- Facilitate the transformation and input of data into a WebSphere Commerce database
- Simplifies loading of large amounts of data
- Four discrete steps
  1. txttransform: transforms a simple character delimited file into a simple XML file
  2. xmltransform: transforms an input XML file, using a user defined XSL Transform, into another file
  3. idresgen: resolves input XML data for input into a database
  4. massload: loads data additions, deletions, updates into the WebSphere Commerce database

To facilitate the discussion of these utilities, a step-by-step example of using the loading utilities is used to illustrate the functionality of these utilities.

The loading utilities allow you to transform and import data into a WebSphere Commerce database. Their benefit becomes apparent when loading large amounts of data into the database, particularly if the data does not contain all the necessary identifiers, such as primary keys.

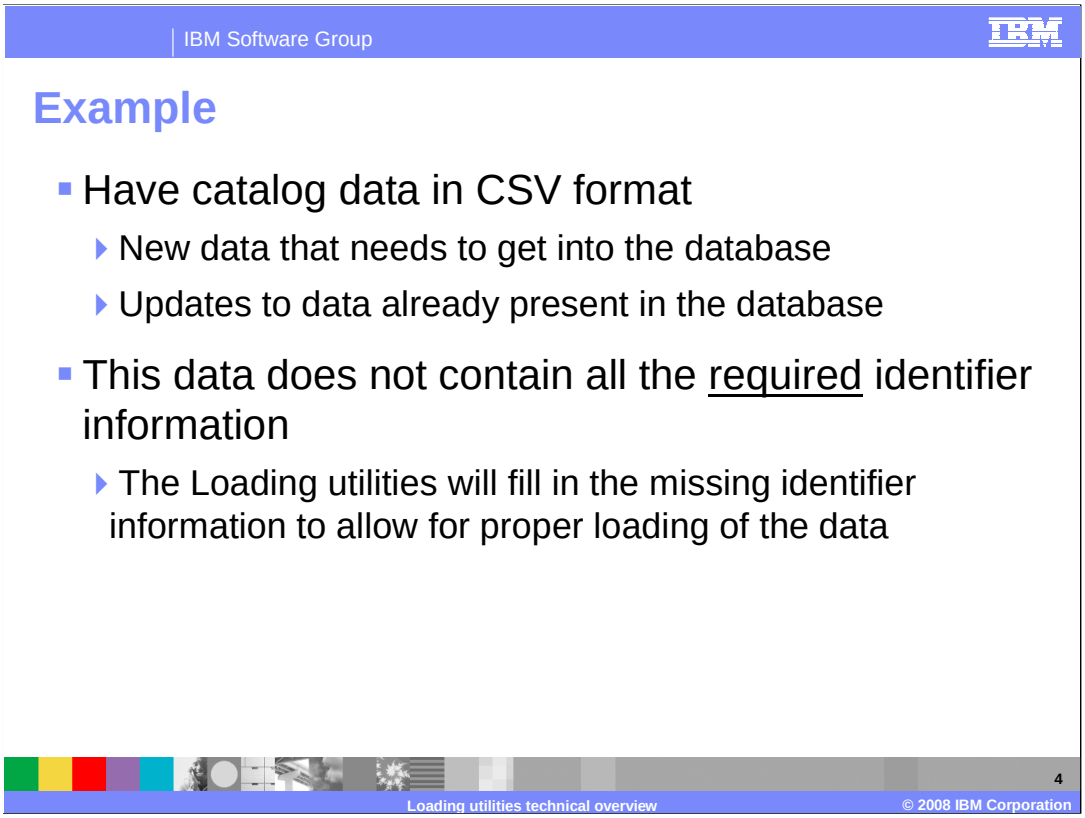
The utilities have been marked one to four in the diagram on the left in the order that they are invoked if using all four utilities in sequence.

The first utility is the txttransform utility. It is able to transform a simple character delimited file into a simple XML file.

The second utility is the xmltransform utility. This utility transforms an input XML file into another file, with the transformation being controlled by a user defined XSL transform.

The third utility is called the idresgen utility, also referred to as the idresolve utility. This utility uses database look-ups to transform a properly formatted XML file into an XML file that is ready for input to the database by the fourth loading utility, the massload utility. The massload utility provides the ability to perform updates, insertions and deletions from a WebSphere Commerce database using an input XML file.

To explain these utilities, an example will be given.



The example that is used is based on the following scenario.

You have CSV data that is associated to catalogs. This CSV data file includes new data that needs to be inserted into the database. It also includes updates to data that is already present in the database and needs to be updated.

This data does not contain all the required identifier information that is needed to directly insert it into the database. It is missing the primary keys that are necessary for this data to properly be used to update existing information, or insert new information into the database.

Example data

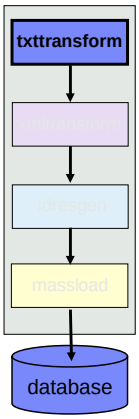
- Contents of initial CSV data file
- One new product and one product to update

	A	B	C	D	E	F	G
1	partnumber	member_id	catenttype_id	markfordelete	available	published	shortdescription
2	SAMPLE_1	7000000000000000002	ItemBean	0	1	1	Sample New Product
3	SAMPLE_2	7000000000000000002	ItemBean	0	1	1	Update Product
4							

In this spread sheet you can see the contents of the CSV data file. It is important to note that a simple data file is being used to illustrate the use of the tools.

The data file contains information about catalog entries. This information is meant to go to two different tables, the CATENTRY table and the CATENTDESC table. In this example, some key values like the CATENTRY_ID are not present.

txttransform utility

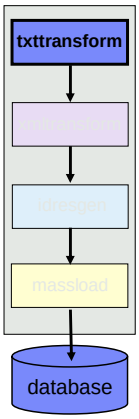


- Convert character delimited file to simple XML file
- Input:
  - character-separated value
  - Schema file defining the format of the XML to transform the data into
- Output:
  - Generated XML file

The first tool in the flow is the txttransform utility.

This utility takes in a file containing character-separated data, and a user defined schema file describing how the plain text file needs to be transformed. Once passed through the txttransform utility, the input file is converted into a well-formed XML file based on the rules in the schema file.

txttransform utility – example schema




```
<?xml version="1.0" encoding="UTF-8" ?>
<TextSchema DataType = "CSV Format">
   <RecordDescription
      FieldSeparator = ","
      RecordSeparator = "&#010;&#013;"
      StringDelimiter = "&quot;"
      HeaderIncluded = "true"
      HeaderLines = "1"
      ElementName = "catentryinfo">

       <FieldDescription FieldName = "partnumber" FieldPosition = "1" />
       <FieldDescription FieldName = "member_id" FieldPosition = "2" />
       <FieldDescription FieldName = "catenttype_id" FieldPosition = "3" />
       <FieldDescription FieldName = "markfordelete" FieldPosition = "4" />
       <FieldDescription FieldName = "available" FieldPosition = "5" />
       <FieldDescription FieldName = "published" FieldPosition = "6" />
       <FieldDescription FieldName = "shortdescription" FieldPosition = "7"/>
   </RecordDescription>
</TextSchema>
```


This is the schema that is used to transform the CSV data.

The section highlighted in red describes the structure of the input file.

The DataType attribute is used to describe the data format of the character-delimited input file.

The RecordDescription tag is where the actual structure of the input file is described.

The FieldSeparator attribute specifies the character, or characters separating fields in the input file. In this case, it is a comma.

The RecordSeperator attribute specifies the character, or characters that separate the records in the input file. It is worthwhile to note that special characters must be entered as a decimal numerical Unicode entity. In this example, a line feed and a carriage return separate records. This is expressed in Unicode as the value that is set in the example.

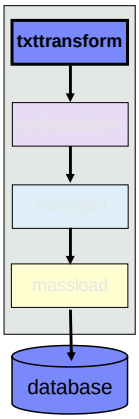
The StringDelimiter attribute refers to the character or characters that enclose each field in a record in the input file.

The HeaderIncluded attribute refers to whether the input file contains the field names as the first line of the input file.

Along side this is the HeaderLines attribute. This is used to tell the transforming tool how many lines in the input file are considered to be part of the header file, and hence not transformed into XML.

Finally there is the ElementName attribute. This attribute provides the tool with the root element name for each of the records.

txttransform utility – example schema




```
<?xml version="1.0" encoding="UTF-8" ?>
<TextSchema DataType = "CSV Format">
   <RecordDescription
      FieldSeparator = ","
      RecordSeparator = "&#010;&#013;"
      StringDelimiter = "&quot;"
      HeaderIncluded = "true"
      HeaderLines = "1"
      ElementName = "catentryinfo">

       <FieldDescription FieldName = "partnumber" FieldPosition = "1" />
       <FieldDescription FieldName = "member_id" FieldPosition = "2" />
       <FieldDescription FieldName = "catenttype_id" FieldPosition = "3" />
       <FieldDescription FieldName = "markfordelete" FieldPosition = "4" />
       <FieldDescription FieldName = "available" FieldPosition = "5" />
       <FieldDescription FieldName = "published" FieldPosition = "6" />
       <FieldDescription FieldName = "shortdescription" FieldPosition = "7"/>
   </RecordDescription>
</TextSchema>
```


The main area of the schema is now highlighted in red.

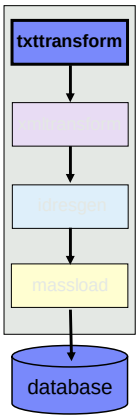
The FieldDescription element is used to describe a field in the input file. It contains the FieldName and FieldPosition attributes.

The FieldName attribute defines the name of the field. This attribute is output as a child element of the root element.

The FieldPosition attribute indicates the position of this field in the record. The first field in a record is at position one.

This schema is used by the txttransform tool to transform the input character-separated file into a well-formed XML file.

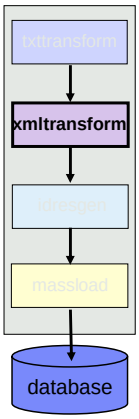
txttransform utility – example output




```
<?xml version="1.0" encoding="UTF-8" ?>
<RootElement>
<catentryinfo
   partnumber="SAMPLE_1"
   member_id="7000000000000000002"
   catenttype_id="ItemBean"
   markfordelete="0"
   available="1"
   published="1"
   shortdescription="Sample New Product"
/>
<catentryinfo
   partnumber="SAMPLE_2"
   member_id="7000000000000000002"
   catenttype_id="ItemBean"
   markfordelete="0"
   available="1"
   published="1"
   shortdescription="Update Product"
/>
</RootElement>
```


This is the contents of the output file that is output from the txttransform utility. As specified by the schema's ElementName field, each record has been associated to a catentryinfo element. The details of the record have been transformed into attributes of the catentryinfo element.

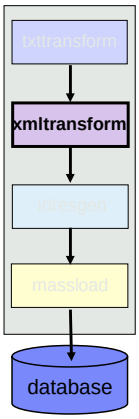
xmltransform utility



- Uses XSLT to transform an XML file into a different format
- Input:
  - Simple XML format
  - XSL transform
- Output:
  - Result of running XSLT against XML format

This leads to the xmltransform utility. This utility uses a user created XSL transform to transform an input XML file into a user defined output file. As input, it takes in a well-formed XML file and the user-developed XSLT and will output a file whose contents are the result of running the XSLT against the input XML file.

xmltransform utility – example (output)



Input Snippet

```
<?xml version="1.0" encoding="UTF-8" ?>
<RootElement>
<catentryinfo
   partnumber="SAMPLE_1"
   member_id="7000000000000000002"
   catenttype_id="ItemBean"
   markfordelete="0"
   available="1"
   published="1"
   shortdescription="Sample New Product"
/>
…
</RootElement>
```

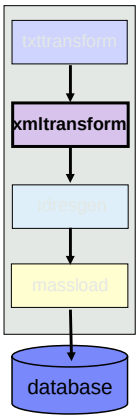
Output Snippet

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE import SYSTEM "wcs.dtd">
<import>
  <catentry member_id="7000000000000000002" catenttype_id="ItemBean"
   partnumber="SAMPLE_1" markfordelete="0"/>
  <catentdesc catentry_id="@SAMPLE_1@7000000000000000002" language_id="-1"
   available="1" published="1" shortdescription="Sample New Product"/>
…
</import>
```


The example shows the xmltransform utility's functionality. The contents of the initial character-separated file were associated with two separate tables in the WebSphere Commerce database, namely the CATENTRY and CATENTDESC tables. The XSL transform created for this example has a goal of separating the data into its appropriate sections.

After running the xmltransform utility against the input file, the section highlighted in red in the input snippet is converted to the format highlighted in red in the output snippet. This is the information that is directly associated with the CATENTRY table.

xmltransform utility – example (output)



Input Snippet

```
<?xml version="1.0" encoding="UTF-8" ?>
<RootElement>
<catentryinfo
   partnumber="SAMPLE_1"
   member_id="7000000000000000002"
   catenttype_id="ItemBean"
   markfordelete="0"
   available="1"
   published="1"
   shortdescription="Sample New Product"
/>
…
</RootElement>
```

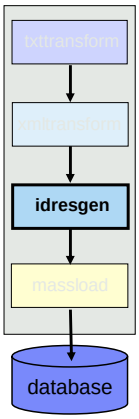
Output Snippet

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE import SYSTEM "wcs.dtd">
<import>
  <catentry member_id="7000000000000000002" catenttype_id="ItemBean"
   partnumber="SAMPLE_1" markfordelete="0"/>
  <catentdesc catentry_id="@SAMPLE_1@7000000000000000002" language_id="-1"
   available="1" published="1" shortdescription="Sample New Product"/>
…
</import>
```


The CATENTDESC information is all that remains. It is transformed by the xmltransform utility into the appropriate output format. This conversion is shown here in red.

Note that the actual implementation of the XSLT file is beyond the scope of this example.

idresgen utility



- Input:
  - XML data file
    - marked up with “@” character for identifier resolution
  - Optional properties file
- Output:
  - XML file
    - Reformatted to contain resolved identifiers

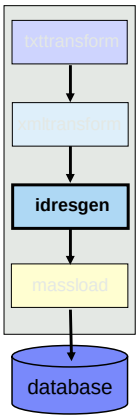
After the initial character-separated file has been transformed into an XML file, it is time to prepare this data for insertion into the database. The required primary keys for getting the data into the WebSphere Commerce database are missing. With large amounts of data, it can be tedious to manually try to figure out the correct values for these required fields. This is where the idresgen utility comes in.

Its primary purpose is to either discover the correct identifiers in the case of information that is already present in the database, or to generate unique identifiers in the case that the data is not already present.

For input, it takes in a well-formed XML data file that can be marked up with the @ character. The purpose of this character is discussed shortly. In addition to the input XML file, the idresgen tool also can optionally accept a custom properties file. This properties file will also be explained shortly. The method parameter for the idresgen command can be given the value load, update or mixed.

The output of the idresgen utility is an updated version of the input XML file with the proper identifiers and their values added.

idresgen utility – input file format



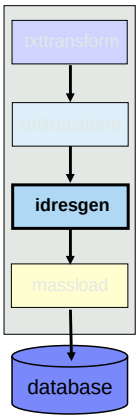
- Three different types of resolution
  - Internal-alias resolution
  - Properties-file specification
  - Unique-index resolution
  - Can be used in conjunction with one another
- "@" based delimiter used for resolution
- KEYS and SUBKEYS table

There are three different types of identifier resolution that are used by the idresgen utility. These include internal-alias resolution, properties file specification and unique-index resolution. Which resolution format to use is based on the input information provided to the utility. Some of these resolution methods can be used in conjunction with one another.

The @ character is used to markup some of the contents of the input XML file to assist idresgen in resolving identifiers.

It is important to note that the idresgen utility only resolves identifiers for a primary table. A primary table must be listed in the KEYS or SUBKEYS table. If it is necessary to resolve identifiers for a table that is not in the KEYS or SUBKEYs tables, add this table to the SUBKEYS table before running the idresgen utility.

idresgen utility – internal-alias resolution (1)



- "@" based delimiter placed in the primary key attribute in the XML input file as an alias
  - scoped to single xml file

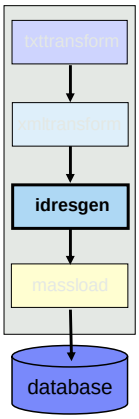
```
<ADDRBOOK
    ADDRBOOK_ID="@addrbook_1" ← Alias for ADDRBOOK
    MEMBER_ID="100"
    DISPLAYNAME="Friends"
    DESCRIPTION="All my friends"
    TYPE="P"
/>
<ADDRESS
    ADDRESS_ID="@address_1"    ← Alias for ADDRESS
    ADDRBOOK_ID="@addrbook_1" ← Refers to the alias for ADDRBOOK
    MEMBER_ID="101"
    NICKNAME="Bob"
    ADDRESS1="1 Brave Developer St."
    CITY="Toronto"
    ZIPCODE="A3B0F4"
    COUNTRY="Canada"
    STATUS="P"
/>
```


For the purpose of explaining each of the forms of resolution, an example snippet from the WebSphere Commerce V6 Information Center is used. These example snippets are not part of the general example being used to tie in all the loading utilities.

This example demonstrates the use of internal-alias resolution. With internal alias resolution, an alias is substituted in place of the primary key or identifier. This alias is then used elsewhere in the XML document to refer to that element. This eliminates the need to know the unique indexes necessary to build the XML file.

In this example, the value for the ADDRBOOK_ID and ADDRESS_ID are unknown, and need to be generated by the idresgen utility. The alias "@addrbook_1" is a unique reference to a unique value for this ADDRBOOK record and the alias "@address_1" is a unique reference to a unique identifier for the ADDRESS record. It is scoped to this file alone, meaning that it will not carry its value over to other files.

idresgen utility – internal-alias resolution (2)



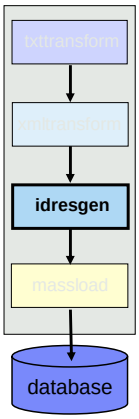
- Output of running input through idresgen
  - Alias replaced by resolved identifier

```
<ADDRBOOK
    ADDRBOOK_ID="11801"     ← Generated primary key
    MEMBER_ID="100"
    DISPLAYNAME="Friends"
    DESCRIPTION="All my friends"
    TYPE="P"
/>
<ADDRESS
    ADDRESS_ID="11901"      ← Generated primary key
    ADDRBOOK_ID="11801"     ← Refers to ADDRBOOK entry
    MEMBER_ID="100"
    NICKNAME="Bob"
    ADDRESS1="1 Brave Developer St."
    CITY="Toronto"
    ZIPCODE="A3B0F4"
    COUNTRY="Canada"
    STATUS="P"
/>
```


After running the input example from the previous slide through idresgen, it is evident how these values were used. All occurrences of the initial “@addrbook_1” alias were replaced with the value 11801, which was generated by the idresgen utility and all the values for the @address_1 alias were replaced with the generated value 11901.

The idresgen utility did not grab these numbers out of thin air. It looked into the KEYS and SUBKEYS table for references of the ADDRBOOK and ADDRESS table to decide how to resolve the values for ADDRBOOK_ID and ADDRESS_ID based on this information.

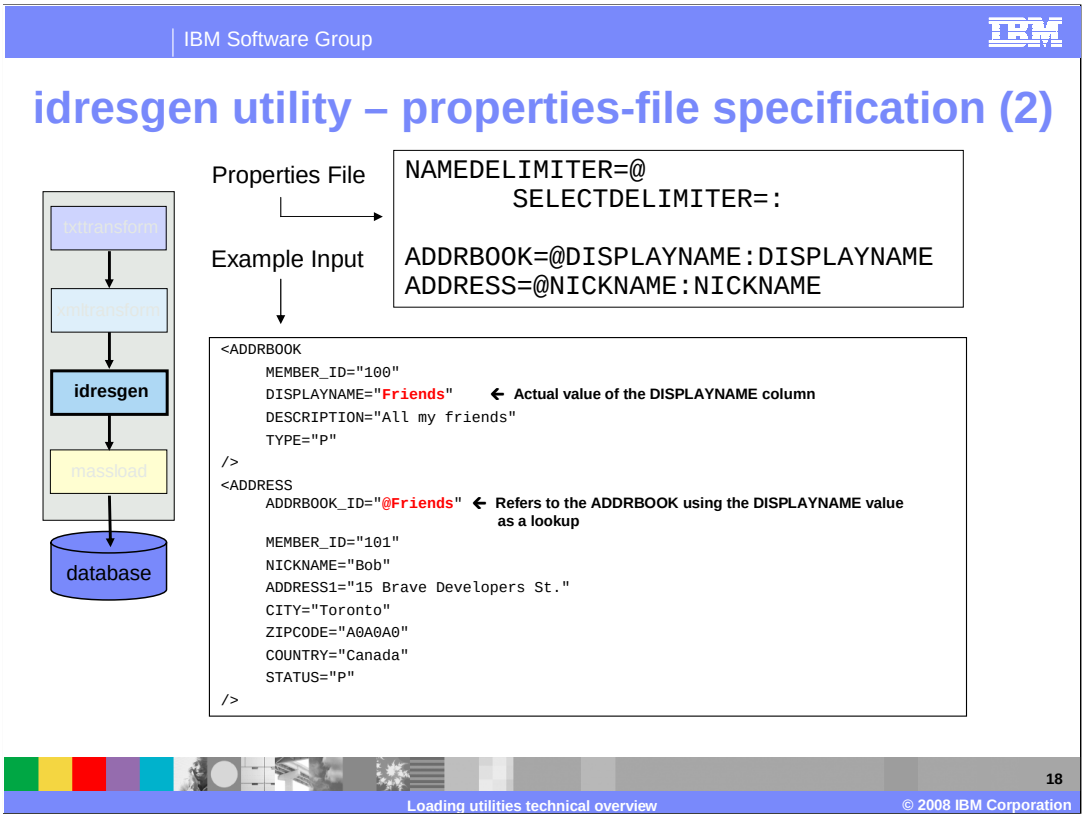
idresgen utility – properties-file specification (1)



- Java style properties file
  - Describes primary table lookup columns
  - Single key
  - Compound key
- Takes precedence over unique-index resolution
- Default file used if no file specified

  WC_installDir/properties/IdResolveKeys.properties

Another form of resolution is through a properties files specification. A Java style properties file can be passed into the idresgen utility as a parameter. This table describes the primary lookup columns. It allows for the declaration of single key lookups and compound key lookups. The idresgen utility gives the property file specification precedence over unique index resolution. If no property file is passed into the idresgen utility, it will use the default one called IDResolveKeys.properties which is located in the properties directory under the base WebSphere Commerce installation directory.



At the top of the screen is an example of the contents of a custom properties file to be used with the idresgen utility.

The NAMEDELIMITER and SELECTDELIMITER set the delimiters used throughout the properties file. These must be used consistently.

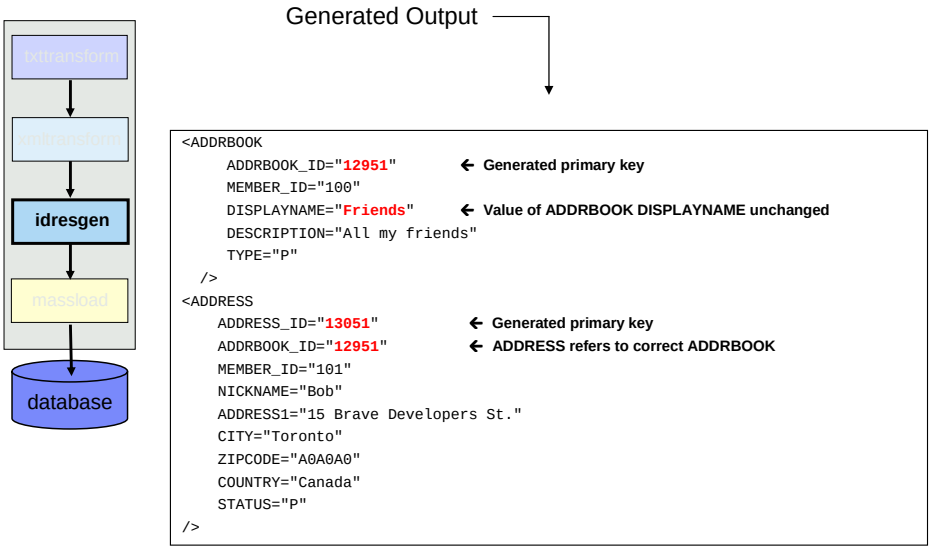
Below that, are the table-value pairs that allow the idresgen utility to figure out how to resolve identifiers.

The first line defines the resolution properties for the ADDRBOOK table. This states that when an address-book record is received, the identifier for the address-book row is created. The DISPLAYNAME field is extracted from the input record and used to form an association to the new identifier. The DISPLAYNAME string is used to match the address-book row DISPLAYNAME and resolve the identifier needed by the foreign key.

At the bottom of the slide, you see some example input for the idresgen utility. As you can see, the DISPLAYNAME for the ADDRBOOK record is “Friends”. This record does not have any primary key identifiers associated to it as those are to be generated by the idresgen utility. Since that reference is not available at the moment, the ADDRESS record uses the DISPLAYNAME value of the ADDRBOOK it wants to be associated with. With the ADDRBOOK_ID value set to “@Friends”, the idresgen utility looks at the properties file to make the connection between the DISPLAYNAME and the ADDRBOOK_ID that it is being used as a reference for.

This property file and input file can be passed into the idresgen utility.

idresgen utility – properties-file specification (3)

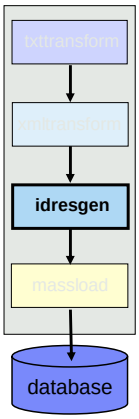


This is the output of the IDResgen utility.

The idresgen tool inserted the ADDRBOOK_ID attribute into the ADDRBOOK element and gave it a value. The idresgen utility did some database lookups to determine what value to assign to ADDRBOOK_ID.

Looking at the ADDRESS element, it is evident that the idresgen not only generated and inserted an attribute for ADDRESS_ID, it replaced the “@Friends” with the generated value of the associated ADDRBOOK_ID element.

idresgen utility – unique index resolution (1)



- Default behavior if:
  - No entries in properties file for table in question
  - No properties file
- Unique index
  - Retrieves identifier for element

```
<CATALOG
     DESCRIPTION="Winter Catalog"
     IDENTIFIER="WC2001"
     MEMBER_ID="100"
     TPCLEVEL="2"
/>
<CATALOGDSC
     CATALOG_ID="@WC2001@100"      ← Refers back to catalog "WC2001" of member "100"
                                      (Note: The order is important.)
     FULLIMAGE="c:\store\img\wc.gif"
     LANGUAGE_ID="-1"
     LONGDESCRIPTION="2001 Winter Catalog"
     SHORTDESCRIPTION="2001 Winter Catalog"
     NAME="B2B direct 2001 Winter Catalog"
     THUMBNAIL="c:\store\img\wc_th.gif"
/>
```

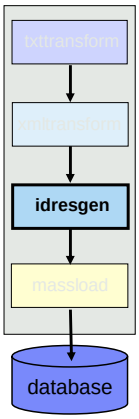

The next form of identifier resolution used by the idresgen utility is unique index resolution. This is the default behavior of the idresgen utility if there are no entries in the properties file for the table in question.

The idresgen utility will use the unique index of a record to retrieve or generate an appropriate value for the identifier.

In this example, you can see that the CATALOG_ID attribute under the CATALOGDSC element has been assigned the value “@WC2001@100”. This is a representation of the unique index for the associated CATALOG element. The “@” sign is the delimiter. The “@WC2001” refers to the IDENTIFIER attribute of a CATALOG record, and the “@100” refers to the MEMBER_ID attribute of a CATALOG record.

Note that the creation order of the unique indexes is important.

idresgen utility – unique index resolution (2)

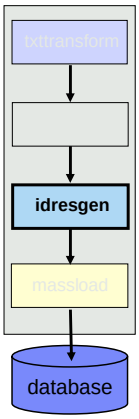


Output file

```
<CATALOG
     CATALOG_ID="10351"                    ←Automatically generated primary key
     DESCRIPTION="Winter Catalog“
     IDENTIFIER="WC2001“
     MEMBER_ID="100“
     TPCLEVEL="2“
/>
<CATALOGDSC
     CATALOG_ID="10351"                    ← Refers to the correct catalog
     FULLIMAGE="c:\store\img\wc.gif“
     LANGUAGE_ID="-1“
     LONGDESCRIPTION="2001 Winter Catalog“
     SHORTDESCRIPTION="2001 Winter Catalog“
     NAME="B2B direct 2001 Winter Catalog“
     THUMBNAIL="c:\store\img\wc_th.gif“
/>
```


Displayed here is the result of running the input file through the idresgen utility. The CATALOG_ID attribute of the CATALOG element was inserted and a value was generated for it. In the CATALOGDSC element, the original value of the CATALOG_ID was replaced with the CATALOG_ID value for the record with the unique index “@WC2001@100”.

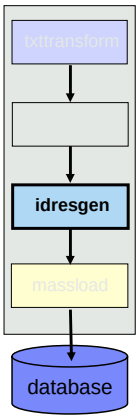
idresgen utility – example (input)




```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE import SYSTEM "wcs.dtd">
<import>
  <catentry member_id="7000000000000000002"
  catenttype_id="ItemBean" partnumber="SAMPLE_1"
  markfordelete="0"/>
  <catentdesc catentry_id="@SAMPLE_1@7000000000000000002"
  language_id="-1" available="1" published="1"
  shortdescription="Sample New Product"/>
  <catentry member_id="7000000000000000002"
  catenttype_id="ItemBean" partnumber="SAMPLE_2"
  markfordelete="0"/>
  <catentdesc catentry_id="@SAMPLE_2@7000000000000000002"
  language_id="-1" available="1" published="1"
  shortdescription="Update Product"/>
</import>
```


Moving back to the main example, it is evident that unique index resolution is being used here. The CATENTRY_ID attributes are being referenced by the unique index for a CATENTRY element.

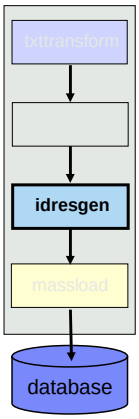
idresgen utility – example (database update)



- No database update required in order to generate unique identifiers
  - CATENTRY row already exists in KEYS table

No database updates were required to generate unique identifiers. Note there is already the required row in the KEYS table for the CATENTRY table.

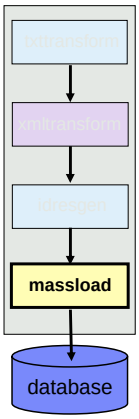
idresgen utility – example (output)




```
<?xml version="1.0"                encoding="UTF-8"?>
…
<import>
<catentry
    MEMBER_ID="7000000000000000002"
    CATENTTYPE_ID="ItemBean"
    PARTNUMBER="SAMPLE_1"
    MARKFORDELETE="0"
    CATENTRY_ID="13451"    />
<catentdesc
    CATENTRY_ID=" 13451"
    LANGUAGE_ID="-1"
    AVAILABLE="1"
    PUBLISHED="1"
    SHORTDESCRIPTION="Sample New Product"                />
<catentry
    MEMBER_ID="7000000000000000002"
    CATENTTYPE_ID="ItemBean"
    PARTNUMBER="SAMPLE_2"
    MARKFORDELETE="0"
    CATENTRY_ID="13452"    />
<catentdesc
    CATENTRY_ID=" 13452"
    LANGUAGE_ID="-1"
    AVAILABLE="1"
    PUBLISHED="1"
    SHORTDESCRIPTION="Update Product"    />
</import>
```


Here is the output of the idresgen utility. Highlighted in red are the attributes that were either updated or inserted into the file by the idresgen utility.

massload utility

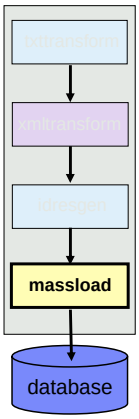


- Data tasks
  - load
  - modify
  - delete
- Input
  - Database information
  - Operation type
  - XML file

Now that the data is in a complete format, it needs to be loaded into the WebSphere Commerce Database. To do this, the massload utility can be used.

The massload utility can be used to load, modify or delete data in a WebSphere Commerce database. For input, it requires the information needed to connect to the database, the operation type that needs to be performed, and the XML file with the data you want to get into the database.

massload utility - input

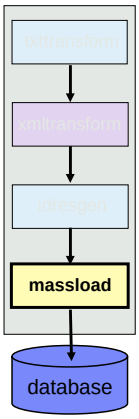


- Input XML
  - Output of idresgen is a valid input
  - Must be valid, well-formed XML
  - Element name → table name
    - Lower case
  - Attribute name → column names
- Referenced DTD is very important
- Outer-most element must be referenced in DTD

The output of the idresgen utility is a valid input for massload. Some rules around this file are; it must contain valid, well-formed XML, the element names correspond directly with the table name, and this must be in lower case. The attribute names correspond to the column names of the table being referenced.

The referenced DTD is very important. It must have all the necessary table elements referenced in it, otherwise massload will not be able to process these missing elements, and will fail. All data in the input XML file must be enclosed in an appropriate element and referenced accurately in the DTD.

massload utility – method parameter



- import
- load
- sqlimport
- delete
- createonly
- loadonly

Listed here are some of the available options for the massload utilities method parameter.

The import method uses the native database import or update functions, if they are available from the database vendor. If the import or update functions are not available, the import method uses SQL statements using JDBC to update the database. If the data being loaded exists in the database, the data is updated with new values from the XML file.

The load method uses the native database loading functions from the database vendor (DB2 Load or SQLLoad). The load method expects your data and the target database to have clean data. This means that the data contains no conflicts or foreign reference problems, and the target database tables do not contain any of the data being loaded. If the data exists in the database, the massload utility fails with a duplicate key error.

The sqlimport method uses JDBC calls to insert and update data for local and remote databases. This method allows column-level updates and allows you to update existing data. The sqlimport method also ensures that records meet the database schema constraints; this makes the sqlimport method safer than the import or load methods.

The delete method is used to delete data that is in the input XML document from the database. The element must contain the values for the primary key or the unique index for the table. If the data being deleted has dependencies to data in another table with "cascade on delete" enabled, the dependent data is also deleted.

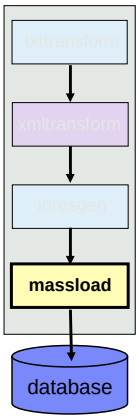
To improve performance during instance creation, use the createonly method. Use the createonly method to create mass load data (MLD) files without loading the data into the database.

Use the loadonly method to load MLD files that were created using the createonly method. When you use the loadonly method, you must also use the -directory parameter.

The WebSphere Commerce Information Center for version 6 contains more detailed descriptions of these method parameters.

massload utility – example

- Run massload using sqlimport method



The final step in the example is to use the massload utility to load the XML data in the WebSphere Commerce database. The method parameter is used with the sqlimport value to ensure that all the database schema constraints are obeyed.

Summary

- Loading utilities are used to simplify the input of data into a WebSphere Commerce database
- Highly extendible and customizable
- Each utility is a stand-alone entity

To summarize, the loading utilities are used to simplify the input of data into a WebSphere Commerce database. They are highly extendible to take into account customizations made to the database. Finally, each utility can be used as a stand-alone entity to accomplish smaller tasks.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_wcs60_LoaderToolsTechnicalOverview.ppt

This module is also available in PDF format at: ../wcs60_LoaderToolsTechnicalOverview.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.