



EVOLUTIONARY
TECHNOLOGIES
INTERNATIONAL

ETI • EXTRACT[®]

*Data System Library Release Notes for
C/SYBASE*

*Release 1.2.3
March, 1998*

ETI•EXTRACT® Data System Library Release Notes for C/SYBASE

Release 1.2.3, March, 1998

THIS DOCUMENT IS THE CONFIDENTIAL AND PROPRIETARY PRODUCT OF EVOLUTIONARY TECHNOLOGIES INTERNATIONAL, INC. ANY UNAUTHORIZED USE, REPRODUCTION, OR TRANSFER OF THIS DOCUMENT IS STRICTLY PROHIBITED. COPYRIGHT © 1997 BY EVOLUTIONARY TECHNOLOGIES INTERNATIONAL, INC. (SUBJECT TO LIMITED DISTRIBUTION AND RESTRICTED DISCLOSURE ONLY.) ALL RIGHTS RESERVED.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Evolutionary Technologies International, Inc.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013 (48 CFR Ch.2) and/or the Commercial Computer Software Restricted Rights clause at FAR 52.227.19(c). Unpublished: Rights reserved under the copyright laws of the United States. Contractor/manufacturer is:

Evolutionary Technologies International, Inc.
4301 Westbank Drive, Bldg. B
Austin, Texas 78746

(512) 327-6994

<http://www.evtech.com>

Evolutionary Technologies, International, ETI•EXTRACT, and the ETI logo are registered trademarks, and Data System Library, AnswerLine, AnswerLink, and MetaStore are trademarks of Evolutionary Technologies International, Inc.

All other product names mentioned herein are for identification purposes only, and may be trademarks and/or registered trademarks of their respective companies and/or institutions.

— *Proprietary and Confidential* —

Property of
Evolutionary Technologies International, Inc.



Serial Number: **EXT-UNIX-122-01-D-SYB-ENG-RN**

Contents

Introduction	5
Contacting ETI	5
Related Documentation	5
Online Help	6
ETI•EXTRACT Documentation Online	6
Tcl man Pages	6
Summary of Enhancements and Changes for Release 1.2.2	7
Changes to the sybase Data Access System Object	7
Updates to Grammar Objects	7
sybase_fork_date_format	7
sybase_fork_substring_delimiter	8
Virtual Source Parts	9
Complex Filter Functions	14
Retrieve Schema now Perl Based	15
Problems Resolved in Release 1.2.2	15
Information Available Since the Handbook was Printed	19
Supported Feature Combinations	19
Support for Single-Step Conversions	20
Code Insertion Points	21
Support for Unit-of-Work Processing	21
Sort Considerations for Unit-of-Work Processing	22
Specification of Tables for UOW	23
Specification of Key Columns for UOW	23
Error-handling for Unit-of-Work Processing	24
Controlling Processing by Using Conversion Properties	24

Default Values of time_vs_integrity	25
Bulk Copy Processing	26
Generation of Client Library Calls	27
The Query Program	27
The Populate Program	27
General Template Changes	28
UNIX Command Files.....	28
New Conversion Properties in Release 1.2.1	28
cs_packetize	28
sybase_cs_version	28
syb_ignore_	
messages	29
sybase_bcp_	
version	29
New Bulk Load Conversion Properties.....	29
bulk_load_options Conversion Property.....	30
New Code Insertion Point.....	31
Representing Delimiters in Strings	31
Representing Delimiters in Templates	32
Real Unit Name and Sybase DB Name Properties.....	32
SQL Commit Limit Property	32
Sybase Server Name Property.....	32
Environment Variable Configuration Requirement	33
Program Execution Command Line Arguments	33
Using NULL Values with the rowxfer Routine.....	33
Using NULL Values with the ct_cursor Routine.....	34
SYBASE Emergency Bug Fixes.....	34
Support for Money Data Type	34
Target Part Filter Requires Correct Schema File.....	34
Documentation Corrections.....	36
data_conv_mode Property	36
Reference to Requirement for ANSI C.....	36

Introduction

These *Release Notes* provide the following information about Release 1.2.2, Release 1.2.1, and Release 1.2.0 of the ETI•EXTRACT Data System Library™ (DSL) for C/SYBASE:

- ❑ A summary of enhancements and changes
- ❑ Information that became available after the handbook was printed
- ❑ Known problems and limitations pertaining to this release
- ❑ Documentation corrections to the *ETI•EXTRACT Data Sytem Library Handbook for C/SYBASE* Release 1.2.

Contacting ETI

Should you encounter difficulty in using the DSL for C/SYBASE, have questions regarding the documentation, or have feedback or suggestions that can help ETI improve the DSL for C/SYBASE, please contact the ETI AnswerLine™. ETI places a high value on your success and fulfilling your needs as a customer.

You can reach ETI AnswerLine personnel at:

Customer Area	Phone	Fax	E-mail
North America	800-856-0416 (toll free) 512-327-6994 ext. 450	512-327-6117	eti.answerline@eti.com
Europe	+44(0)118-977-1221	+44(0)118-977-9800	eti.answerline@eti.com

ETI's World Wide Web address is <http://www.eti.com>. You can access the AnswerLink™, ETI's online support center, from our web page.

Related Documentation

Related documentation for the *Data System Library Release Notes for C/SYBASE* consists of the following:

- ❑ The *ETI•EXTRACT Data System Library Installation Guide* provides information about installing ETI•EXTRACT DSLs.
- ❑ The *ETI•EXTRACT Data Sytem Library Handbook for C/SYBASE* contains detailed information about using the DSL for C/SYBASE. It is a supplement to the *ETI•EXTRACT Conversion Specialist's Guide, Master User's Guide, and Reference Manual*.

Note: A README PDF document may be installed with the other PDF files for the DSL. If so, the README document contains information that became available after the *Data System Library Release Notes for C/SYBASE* was printed.

Online Help

The online help for ETI•EXTRACT is available by using the Help menu in any ETI•EXTRACT tool window, or by pressing the **F1** key. You can access general information by clicking **Help** on the Menu Bar. If you want help relative to a task you are performing, press **F1** for context-sensitive help.

ETI•EXTRACT Documentation Online

All of the ETI•EXTRACT documentation is available online in the form of Adobe™ Acrobat Reader™ Portable Document Format (PDF) files. They are located in the `doc` directory of your ETI•EXTRACT installation directory. The ETI•EXTRACT distribution tape includes a copy of the UNIX version of Acrobat Reader, and you can display any of the manuals by using the following command:

```
ex_manual manual_name
```

Substitute the name of a manual (the PDF file name) for *manual_name*.

Entering the **ex_manual** command without the *manual_name* argument displays a list of available manuals.

Tcl man Pages

You can extend the capabilities of ETI•EXTRACT by writing Program Generator Extension Functions and Grammar Extension Functions in Tool Command Language (Tcl). Tcl is a publicly available interpreted language and is embedded in ETI•EXTRACT. For more information on Tcl, refer to *Practical Programming in Tcl and Tk*, Brent B. Welch, ISBN 0-13-182007-9.

In addition, ETI•EXTRACT includes UNIX manual (man) pages for Tcl. You can view them by entering the command:

```
man command_name
```

where *command_name* is a Tcl command.

For example, you can display a summary of Tcl language syntax by entering the command:

```
man Tcl
```

Summary of Enhancements and Changes for Release 1.2.2

The following list summarizes the enhancements and changes for Release 1.2.2 of the DSL for C/SYBASE:

- ❑ New DAS type, **text**, has been defined
- ❑ The `date_format` property has been modified
- ❑ New `bcp_delimiter` property has been added
- ❑ New grammar forks `sybase_fork_date_format` and `sybase_fork_substring_delimiter` have been created
- ❑ Support for Source Virtual Parts has been added
- ❑ Support for complex filter functions is now available
- ❑ The Retrieve-Schema mechanism is now Perl Based

Changes to the sybase Data Access System Object

The following changes have been made to the **sybase** Data Access System (DAS) object for Release 1.2.2 of the DSL for C/SYBASE:

- ❑ A new data type **text** which corresponds to the ETI•EXTRACT type `varstring` was added.
- ❑ Two data types were updated:
 - When you set the DAS type to bit, the ETI•EXTRACT type is set to **string** (instead of integer)
 - When you set the DAS type to text, the ETI•EXTRACT type is set to **varstring**
- ❑ The property **date_format** has been changed from `MMM DD YYYY HH:MIAP` to **0**
- ❑ The property **bcp_delimiter** has been added with a default value of |
- ❑ A new attachment has been added:
 - Grammar Name = `sybase_new_source_part.1`
 - Grammar Type = Create Source Part

Updates to Grammar Objects

The following changes have been made to the Grammar objects for Release 1.2.2. of the DSL for C/SYBASE. Two new grammars were added:

- ❑ `sybase_fork_date_format`
- ❑ `sybase_fork_substring_delimiter`

`sybase_fork_date_format`

A new source and target part grammar fork, `sybase_fork_date_format`, queries a date format and stores it in the conversion property `date_format`. It provides a point-and-click interface for setting the conversion property to a valid numeric value.

sybase_fork_substring_
delimiter

A new target part grammar fork, `sybase_fork_substring_delimiter`, allows you to specify delimiting characters or substrings for processing the value of the target part. Rather than setting the modify-value buffer in a populate program to the value of the target part, you can apply a filter to return a substring based on the specification of delimiting characters or substrings. (This integrates the consultant package `CP_syb_120_tgt_substr`.)

Note: Release 1.2.1 of the DSL for C/SYBASE supports only integer constants and global variable arguments in the substring function.

Applying a target part filter grammar to create modify-value text now lists an option for delimiting characters or substrings in the substring of a target part.

You can key on a position represented by the location of a specified character string. You are prompted for the first and last characters in the string, and the delimiters (before and after) of the character string.

The Filtering window lists the following options:

- character before the delimiter
- first character of the delimiter
- character after the delimiter
- last character of the delimiter

End the string with an integer value, a global variable, a manifest constant, or as determined by the position of another delimiting string.

Example In Release 1.2.1, the following statement could be used:

```
mod_val_buf = subseq(INFILE1_struct_ptr->mycolumn, 2, 5)
```

You can now use the following statement for Release 1.2.2 to select a substring from the string:

```
mod_val_buf = subseq(INFILE1_struct_ptr->mycolumn,  
                    getindex(INFILE1_struct_ptr->mycolumn, 'x', 1, 1),  
                    getindex(INFILE1_struct_ptr->mycolumn, 'z', 1, 1)  
                    );
```

Create modify-value text for the target part `mycolumn` as follows:

- Return the substring of `mycolumn` starting at the index position determined by string 'x'.
- Use the specified occurrence of 1.

- ❑ The first character of the string is the delimiter.
- ❑ Include the index string 'z'.
- ❑ Use the specified occurrence of 1.
- ❑ The first character of the string is the delimiter.

If the value of the target part were *xyzz*, the resulting value of the character string is **xyz**.

Using the following statement:

```
mod_val_buf = subseq(INFILE1_struct_ptr->mycolumn,
                    getindex(INFILE1_struct_ptr->mycolumn, "z", 2, 1),
                    getindex(INFILE1_struct_ptr->mycolumn, "y", 1, 1)
                    );
```

the value of the character string is **zy**.

Virtual Source Parts

The DSL for C/SYBASE provides extended grammar support for creating virtual source parts. A *virtual part* is a part that you create in a conversion specification which can be used to compute the value of another real or virtual part or to map to a target part. After a virtual part is created, it is listed in the Conversion Editor. A **V** character appears within a circle in the part icon, indicating that it is a virtual part.

 **newvirtpart**

The scope of a virtual part is the conversion in which you create it. For example, if you create a virtual part for the TEST conversion, that virtual part is available in the TEST conversion only. A virtual part does not exist in the source database and creating it does not change the schema of the source database.

Virtual parts are created as non primary keys. You can assign a value to a virtual part as follows:

- ❑ Specify a default value or expression when you create the virtual part.
- ❑ CIP to populate the virtual part.

The following steps describe how to create a virtual source part.

1. Select a source unit or part. If you select a unit, the virtual part is placed at the end of the parts list.
2. Select **Conversion: Add Virtual Part**. The Add Virtual Part window appears, as shown in Figure 1.



Figure 1. Adding a Virtual Part

3. Specify the number of new virtual parts you want to create. If you create multiple virtual parts in the Add Virtual Part(s) window, they are given unique names based on the part name that you enter. They also have the same attributes as the initial virtual part. You can modify their names and attributes in the Conversion Editor after you create them.
4. If you select a source part in the Conversion Editor, specify where you want the virtual part added in the parts list (**At end**, **Before selected item**, or **After selected item**).
5. Click **OK**. The Filter window opens to create a virtual source part, as shown in Figure 2. This window has most of the same options as the Filtering window used to apply a filter.

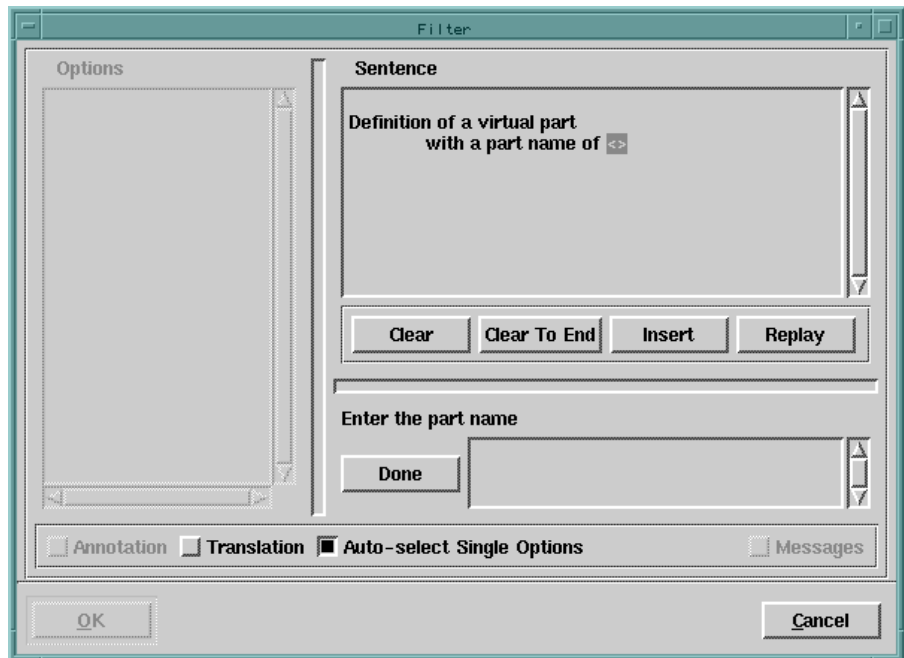


Figure 2. Filter Window for Creating a Virtual Source Part

6. Enter a name for the virtual part and click **Done**. An Options list of ETI•EXTRACT data types opens, as shown in Figure 3.

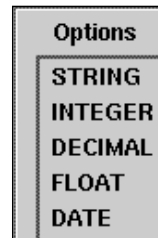


Figure 3. Data Types

7. Select the ETI•EXTRACT data type for the virtual part.
8. Specify the length for the virtual part. The options displayed depend on which data type you selected.

Note: Do not include the sign or decimal point as part of precision values; ETI•EXTRACT adds required items such as signs and decimal points to precision values. If you select a default length, signs and decimal points are automatically included.

Table 1. Data Type Options for Virtual Parts

Data Type Selected	Action
STRING	Enter the length; then, click Done .
INTEGER	<p>Select one of three options:</p> <ul style="list-style-type: none"> • using a default length of 39 • using a default length of 11 (max for COBOL) • specifying a precision of <p>If you select specifying a precision of, enter the precision value and click Done. The default lengths of 39 and 11 are provided for compatibility with the maximum lengths supported by the sybase DAS and the COBOL language, respectively.</p>
DECIMAL	<p>Select one of three options:</p> <ul style="list-style-type: none"> • using a default length of 40 • using a default length of 20 (max for COBOL) • specifying a precision of <p>The default lengths of 40 and 20 are provided for compatibility with the maximum lengths supported by the sybase DAS and the COBOL language, respectively.</p> <ul style="list-style-type: none"> • <i>If you select a default length of 40 or 20, you are prompted to enter the length of the digits to the right of the decimal point. The remainder are used for the digits to the left of the decimal point.</i> • <i>If you select specifying a precision of, you are prompted to enter the length of the digits to the left and right of the decimal point.</i> <p>Click Done after entering the value(s).</p>
FLOAT	<p>Select one of three options:</p> <ul style="list-style-type: none"> • using a default length of 46 • using a default length of 22 (max for COBOL) • specifying a mantissa precision of <p>The default lengths of 40 and 20 are provided for compatibility with the maximum lengths supported by the sybase DAS and the COBOL language, respectively. If you select specifying a precision of, you are prompted to enter the length of the digits in the mantissa and click Done.</p>
DATE	The default length of 26 is assigned. If you want to change the length, modify the Length column of the virtual part after you create it.

The Options list to specify a default value opens.

9. Select either **a default value of** or **no default value**.
The list includes the following default value options:

a default value of

If you are creating a string part, a subsequent Options list opens, as shown in Figure 4. To specify that the default value you enter is delimited by apostrophes, select **a value**; otherwise, select **open text**. Open text lets you enter such text as SQL functions and their arguments. Use delimiters where necessary in open text.

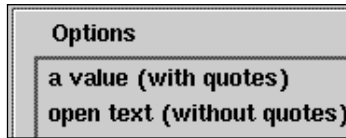


Figure 4. Default Value Options for String Virtual Parts

For a non string part, enter either a literal value or an SQL function. Use delimiters where necessary.

Enter the default value and click **Done**. The default value you enter is used in an SQL SELECT statement to assign an initial value to the virtual part.

sql_default_value

The context variable is set when you specify a default value.

no default value

String parts are assigned spaces to the length of the part. Numeric parts are assigned a value of zero.

The options for entering a documentation note appear next.

10. Select **a documentation note of** to document the virtual part. Enter the documentation note and click **Done**.

The text you enter appears in the **Annotation** pane of the Schema Properties window for the virtual part. To view the annotation later, select the virtual part in the conversion specification; then, select **Schema: Schema Properties**.

Select **no documentation note** to skip documentation.

11. Click **OK** to create the virtual part and exit the Filter window.

After you create a virtual part, you can modify its attributes in the Conversion Editor by editing its conversion properties.

To delete a virtual part, select the part in the Conversion Editor; then, select **Edit: Delete**.

Complex Filter Functions

Prior to C/SYBASE 1.2.2, SQL Filters could be applied to source parts by applying a source part filter and choosing the following path:

```
<<New Filter>>
```

```
SQL Filters
```

```
Apply function
```

The user could then choose either a complex function or as many simple functions as desired optionally ended by a complex function. The point being that any number of simple functions could be applied but once a complex function was chosen, no more functions could be chosen for that part.

For example, the above path could be further followed by these choices:

```
lower
```

```
reverse
```

```
ltrim
```

```
substr
```

```
starting at position 1
```

```
for a length of 2
```

whereupon the following sample code would appear in the SQL command being built:

```
lower(reverse(ltrim(substring(THIS_UNIT.THIS_PART, 1, 2))), \
```

and no further functions could be applied after having chosen substr as it is implemented as a complex function, i.e., it takes more arguments than just the part name.

The simple functions supported by `sybase_fork_sql_fcn` were as follows:

```
lower, ltrim, rtrim, avg, count, max, min, sum, upper, reverse
```

The complex functions supported by `sybase_fork_sql_fcn` were as follows:

```
avg distinct, sum distinct, substr, lpad, replace, rpad,  
translate
```

However, with C/SYBASE 1.2.2, no such distinction is made between simple and complex functions such that any number of SQL functions can be chosen and the list built in the SQL command for the source part will only be terminated when the user specifies that no other SQL functions are to be applied, for example, to generate the following:

```
LOWER(REVERSE(SUBSTRING(LTRIM(THIS_UNIT.THIS_PART), 1, 2))), \
```

the user would make the following choices:

```
<<New Filter>>
```

```
SQL Filters
```

```
Apply function
```

STRING function
 LOWER
 and SQL function
 STRING function
 REVERSE
 and SQL function
 STRING function
 SUBSTR
 starting at position 1
 for a length of 2
 and SQL function
 STRING function
 LTRIM

whereas such a path was not possible prior to C/SYBASE 1.2.2.

Retrieve
 Schema now
 Perl Based

The retrieve-schema mechanism has been rewritten from a grammar-based tool to a perl-based tool.

Problems
 Resolved in
 Release 1.2.2

Table 2 describes the reported problems resolved in Release 1.2.2 of the DSL for C/SYBASE. They are listed by Product Design Problem Report (PDPR) number.

Table 2. Product Design Problem Reports Resolved in Release 1.2.2

PDPR	Description
1985	The Grammar sybase_fork_data_conv_mode (provided to support setting <code>data_conv_mode</code>) has been modified such that the candidates that are listed are not longer limited to string parts, but include those parts that correspond to the appropriate data-type. (Appropriate data-type is defined by the previous choices that you have selected in the filter window.)
2047	The Grammar sybase_fork_sql_default_value (provided to support setting <code>sql_default_value</code>) has been modified such that it will automatically check for both string and numeric parts, defaulting such parts to spaces or zeroes respectively (rather than requiring the user to choose between the two values).

Table 2. Product Design Problem Reports Resolved in Release 1.2.2

PDPR	Description
2925	<p>Program calls to errprt with more than one argument now reference append_error. The following templates were changed to reference append_error rather than passing more than one argument to errprt:</p> <ul style="list-style-type: none"> • ERROR-FILE-SUPPORT • define-ifile-open • fcn-curr-datetime
3028	<p>The template generate-bcp now generates an error message if you don't set data_conv_mode to insert when enabling BCP.</p>
3404	<p>You can now assign a default value to unmapped date parts. The template COLUMN-DEF-DFLT now references sql_default_value in addition to pop_default_value.</p>
4511	<p>Spelling errors in the messages generated by the retrieve-schema have been corrected.</p>
4903	<p>The template process-ifile-for-target-unit correctly reports the record number by referencing record_count (instead of commit_interval). Messages beginning with At Record number in the .wrn file now list the correct record number.</p>
5251	<p>The template format-ifile-float-part processes decimal values properly by referencing floating point parts with an f (rather than an E).</p>
5329	<p>The template part-write-single-step no longer stores numeric parts which are NULL in intermediate Buffer 2: the input work area.</p>
5515	<p>Retrieve_Schema parses text data-types:</p> <ul style="list-style-type: none"> • The grammar extension module DSL-create-fcns was changed so the lst_sybase_datatypes function now accepts text data type. • The grammar extension module DSL-RS-fcns was changed so that the fmt_sybase_datatypes function now accepts text data type. • The following templates were changed: <ul style="list-style-type: none"> • INSERT-INTO-TABLE-BCP • UPDATE-WHERE-CLAUSE • sybase-cursor-string • The sybase DAS now has a type text, that corresponds to the ETI•EXTRACT type of varstring.
5534	<p>The templates check-for-populate and check-for-query now correctly generate the program and file names within the shell scripts.</p>
5535	<p>The template POPULATE has been modified to include the code insertion point (CIP) post-populate only once when prog_type is set to bulk_load (which enables bulk loading from flat files).</p>

Table 2. Product Design Problem Reports Resolved in Release 1.2.2

PDPR	Description
5537	<p>You can now apply a modify-value filter to a target part to check for NULL values and to specify different values with the following templates and grammar:</p> <ul style="list-style-type: none"> • filt-part-string-non-repeat-setting • filt-memcpy-for-bulk-load • filt-part-string-setting-len • sybase_c_modval (grammar)
5719	<p>The template activate-transaction-mode now executes commits in compliance with sql_commit_limit when prog_type is set to bulk_load (which enables bulk loading from flat files).</p>
6185	<p>The data types of the ETI•EXTRACT DAS type bit is written as a string of one character (not an unsigned integer).</p>
6222	<p>A new grammar fork, sybase_fork_date_format, supports data formatting when filtering source parts. The grammar sybase_sybase_src_part was changed. The following templates support date formatting on the source side:</p> <ul style="list-style-type: none"> • SQL-PART-CURSOR-REAL • convert-date-part
6258	<p>Applying a target filter to compare a part for spaces now checks for all spaces, not just one space.</p>
6430	<p>Reference to cip_cursor_sql_end_cb was changed to cip_sql_cursor_end_cb.</p>
6590	<p>The Grammar Extension Module DSL-create-fcns has been modified such that the lst_sybase_datatypes function now processes the data-types intn, nchar, nvarchar, and moneyn. The Grammar Extension Module DSL-RS-fcns has been modified such that the fmt_sybase_datatypes function now processes the data-types intn, nchar, nvarchar, and moneyn. The Templates INSERT-INTO-TABLE-BCP, UPDATE-WHERE-CLAUSE, and sybase-cursor-string have been updated such that the data-types intn, nchar, nvarchar, and moneyn are now processed.</p>
6669	<p>Filter concatenation no longer frees global variables.</p>
6706	<p>A new template, tpl-cip-define-prototypes, was added to prototype custom functions before rather than defaulting to integer functions. The template BASE-QUERY was modified to include references to tpl-cip-define-prototypes and tpl-cip-fcns to use the following Code Insertion Points (CIPs) in the generated code:</p> <ul style="list-style-type: none"> • cip_define_prototypes • cip_define_prototypes_cb • cip_fcns • cip_fcns_cb
6710	<p>The server name prompt was corrected.</p>
6791	<p>References to current-target-part in DSL-filter-fcns were resolved.</p>

Table 2. Product Design Problem Reports Resolved in Release 1.2.2

P DPR	Description
6967	Numeric values and parts are now treated distinctly when specifying auxiliary table lookups. The grammar <code>sybase_fork_aux_xref</code> (used for setting auxiliary table lookups) now treats numeric values different than numeric parts.
6781	ORACA and SQLCA calls have been removed from the template code
6972	The Template module <code>sybc_com/base/single,multi-close-conn-at-logoff</code> has been modified to check for a value not equivalent to <code>CS_SUCCEED</code> instead of <code>CS_PENDING</code> .
6973	The Template module <code>syb_sybc_pop_ctlib/aux-function</code> has been modified such that the <code>CS_COMMAND</code> variable referenced as <code>ct_cmd_alloc(connection, &cmd)</code> is now referenced as <code>aux_cmd</code> rather than <code>cmd</code> .
6974	The Template module <code>syb_sybc_pop_ctlib/base/aux-function</code> has been modified so that all calls to <code>ct_cmd_alloc</code> have a corresponding call to <code>ct_cmd_drop (cmd)</code> .
7006	The <code>data_conv_mode</code> of <code>data_driven</code> now works correctly. (The Template module <code>syb_sybc_populate/base/data-driven-mode</code> has been modified to process <code>update_or_insert</code> properly when <code>data_conv_mode</code> is set to <code>data_driven</code> .)
7080	The inaccurate code in the Template modules <code>syb_sybc_cips/tpl-cip-pre,post-ifile-read</code> has been removed.
7087	The function <code>sybase_c_modval</code> now returns the result of applying string, integer, decimal, and float functions with any number of arguments. (The Grammar <code>sybase_c_modval_float_fork</code> was modified to clarify the rule which selects mapped source parts.)
7100	Support for source virtual parts has been added.
7101	You can now set the default target value of <code>SYSDATE</code> . (The Grammar <code>sybase_fork_date_format</code> has been added.)
7122	The Template modules <code>syb_sybc_pop,qry_ctlib/base/program-init-sybase-declare</code> have been modified to reference <code>sybase_cs_version</code> (rather than an explicit <code>CS_VERSION_100</code>).
7124	SQLCA references have been removed from the templates and no longer appear in the generated code.
7148	All Template modules which use variable arguments (varargs) now use both <code>va_start</code> and <code>va_end</code> functions. (<code>va_end</code> provides the clean up process in which resource allocations are freed up.

Information Available Since the Handbook was Printed

This section contains information that was made available after Release 1.2 of the *ETI•EXTRACT Data Sytem Library Handbook for C/SYBASE* was printed. This section contains information for Release 1.2.0 and Release 1.2.1.

Supported Feature Combinations

The following describes combinations of features which are supported in Release 1.2.1 of the DSL for C/SYBASE, and combinations of features which are currently not supported. Table 3 provides a matrix showing combinations of features. The table lists the following features:

- SS Single-Step program generation
- TL Auxiliary Table Lookup
- RPQ Rows to Process while Querying
- UoW UOW
- BCP Bulk Copy Processing
- BL Bulk Loading

The numbers shown in Table 3 indicate the number of simultaneous server connections used to implement the feature. For example, two server connections are used to implement single-step processing. An entry of NS in a table cell indicates that a feature combination is not yet supported.

Table 3. Supported Feature Combinations

	SS		
TL	2	TL	
RPQ	2	1	RPQ
UoW	NS	1	1
BCP	NS	2	1
BL	2	1	1

It is important to note combinations which do not appear in Table 3 as well as those that do. For example, BCP and BL cannot be combined because they require different methods of bulk loading. Bulk Copy Processing generates programmatic bulk loads, while Bulk Loading generates UNIX scripts that act on flat files. Less obvious missing combinations may be UoW + BCP and UoW + BL, which would both involve mutually exclusive combinations. They are not possible because SYBASE prohibits BCP calls in a multi-statement transaction, but that is what is used to implement UOW processing (UoW generates a BEGIN

TRANSACTION statement). Bulk Loading requires that each target unit be acted on individually, while the DSL's implementation of UoW processing requires multiple units to be acted on simultaneously.

With these inappropriate combinations in mind, the additional compound combinations shown in Table 4 are possible.

Table 4. Additional Feature Combinations

Implemented With One Connection	UoW + TL + RPQ TL + BL + RPQ
Implemented With Two Connections	SS + TL + BL SS + TL + RPQ SS + BL + RPQ BCP + TL + RPQ
Not Implemented - Requires Three Connections	SS + TL + BCP SS + TL + BCP + RPQ
Not Yet Implemented	BL + SS + TL + RPQ UoW + SS + RPQ UoW + SS + TL UoW + SS + TL + RPQ SS + BCP + RPQ

Support for Single-Step Conversions

In addition to the templates used to generate query and populate programs, the Data System Library for C/SYBASE also includes templates that support the generation of a single program that combines all conversion program steps. Combining processing into a single conversion program eliminates the I/O operations normally required to create intermediate files that are passed to intermediate programs and to the final populate program.

The following restrictions apply to single-step conversion programs:

- Both the source and target data access systems must be the same.
- The language used to access both source and target data access systems must be the same (C in the case of the DSL for C/SYBASE).
- Both the source and target data access systems must reside on the same host.
- The Data System Library being used must support single-step conversion programs (the DSL for C/SYBASE does provide this support).

If all of these conditions apply, you will find that using a single-step conversion program offers two advantages over performing a multi-step conversion: faster throughput and lower disk space requirements.

To generate a single-step conversion program, set the value of the conversion property **single_step** to true for the conversion.

Code Insertion Points

The code insertion points (CIPs) provided in the DSL for C/SYBASE are listed in Table A-1 in Appendix A of the Data System Library Handbook for the DSL.

Support for Unit-of-Work Processing

The DSL for C/SYBASE includes a Perl script that you can use to define a Unit of Work (UOW) for ETI•EXTRACT and templates that generate the code necessary to process a UOW. The uow Perl script is located in the following directory:

```
SEXTRACT_ROOTDIR/MetaStore/database_name/das/sybase.major/uow
```

Use the following command to parse a text file to define a UOW:

```
perl construct_uow_array.pl [-h] <input> <output>
```

where

[-h] is optional for displaying additional help

<input> is the UOW input file

<output> is the file used to store the output structure

The input file must be formatted as shown below.

```
UOW: 1
TABLE: name1
KEY: key1: 1
KEY: key2: 3
TABLE: name2
KEY: key1: 1
KEY: key12: 2
UOW: 2
TABLE: name3
KEY: key3: 19
```

The colon (:) must follow the label without a space. The UOW label identifies the record as such and its value should be a sequential number. Set the first UOW to begin at one (1), the next UOW should be two (2), and so on. Set the table label value to the name of the table. The key label's value should contain two bits: the first bit is the name of the key, and the second bit is the key's column position in the table (not currently used).

The script imports this information and creates an ETI•EXTRACT structure that the template use to correctly process the transaction.

Unit-of-work processing means coordinating the I/O into the target so that all information pertaining to a single business entity or transaction is committed simultaneously. For example, multiple records and/or record types of payroll data are processed in order by department and committed into the target database only when the value of the department number changes on the incoming intermediate files.

Sort
Considerations for
Unit-of-Work
Processing

For this to work, it is essential that all related data arrive in sorted order. If the data is not sorted or sorted incorrectly, the condition is fatal and the following message is generated:

```
Error: ETI-SYB-E-0039 The data is not sorted
correctly, and I/O into the related tables can not be
coordinated as specified for the
Unit-of-Work #n
```

Two mechanisms accomplish the sort, one for single-step programs, another for multi-step.

Single-step
programs

The DSL generates an `order by` clause on the appropriate `ctlib` call in the query program, when the query cursor is mapped to a table involved in a UOW. It uses the source column name(s) mapped to the key(s) specified for the UOW.

Multi-step programs

The DCT user creates a CVAR on the target unit, named `sort_before_populate`, having as its value the target column names that are involved as keys in controlling the Unit-of-Work processing. ETI•EXTRACT will sort the ifile on all the data mapped to those key parts. In the CVAR, the part names are specified exactly as they appear in the DCT, and are separated by spaces.

Effects of
multi-sourcing a
unit-of-work key
part

When more than one source part is mapped to a target part that is a key for the UOW, the order in which the parts are multi-sourced becomes the order of precedence for the sort. For example, first-name and lastname are mapped to target full-name. Full-name is the key for the UOW. If the correct sort is last-name/first-name, then the DCT user must map the last-name followed by the first-name.

Avoiding an
unnecessary sort

There may be rare circumstances, such as mapping one source table to multiple target tables, that comprise a UOW when the data is naturally in the correct order without requiring a sort. The CVAR `bypass_order_by` allows a DCT user to avoid generating an `order by` clause in single-step programs. In multi-step, if the user fails to specify the CVAR `sort_before_populate`, then the ifile will not be sorted.

Specification of
Tables for UOW

Relative position of the table specifications in a UOW is essential to proper functioning. When more than one table shares the same key data values, they are loaded in the order in which they are specified. If referential integrity constraints are enforced during the populate, it will fail unless dependencies are reflected correctly in the position of tables in the UOW.

For example, assume the following:

1. The intent is to commit each customer's account data, separately from other customers.
2. There are multiple accounts per customer, and ACCOUNT.CUST_NO is a foreign key referencing CUSTOMER.CUST_NO.

Then the following specification will fail:

```
UOW:1
TABLE: ACCOUNT
KEY: CUST_NO : 1
TABLE: CUSTOMER
KEY: CUST_NO : 1
```

But this specification will work correctly:

```
UOW:1
TABLE: CUSTOMER
KEY: CUST_NO : 1
TABLE: ACCOUNT
KEY: CUST_NO : 1
```

Specification of
Key Columns for
UOW

The key columns for a UOW is typically the primary key or a candidate key for the first table, and the foreign key columns reference the UOW for the other tables. Multiple keys are supported, as long as the number of key columns is the same for each table.

```
UOW: 1
TABLE: t-name1
KEY: col-name-1 : 1
KEY: col-name-2 : 2
TABLE: t-name2
KEY: col-name-3 : 1
UOW: 2      <-- need to specify a 2nd key column for t-name2
```

The idea is that the populate program will read the related data in sorted order and match the key data among all the related ifiles. Unmatched data values are processed without exception. (Referential integrity and other constraints may cause a failure of I/O into the target database, but ETI•EXTRACT does not attempt to anticipate that.) Matching occurs based

on concatenated key values, so the order in which multiple keys are specified is crucial. Multiple keys must be specified in their order of precedence, reflected by their position in the UOW specification file.

The numbers at the right end of the key specification are not used at all, but are required for the file to be processed correctly.

Error-handling for
Unit-of-Work
Processing

During the processing of a UOW, the key set for each row processed is stored in an internal table, along with any error messages generated. When the UOW is complete and errors have been encountered, the UOW is rolled back, and the table is dumped into the warning file and then cleared in readiness for the next UOW. If the UOW completes without errors, it is committed and the table is simply cleared for the next UOW.

The internal table structure used to hold the key sets for a UOW is initially specified as containing 50 rows. This can be overridden by defining a context variable called `UoW_Err_Table_Rows`. It is very important that you estimate the potential maximum size of the UOW and allow a sufficient number of rows to hold all key sets and error messages for the largest occurrence of the largest UOW processed; otherwise, the conversion will fail.

Controlling
Processing by
Using
Conversion
Properties

The DSL for C/SYBASE employs the following conversion properties to control how programs generated by the DSL process data:

row_count Set on a source database. This value denotes the number of rows to read at a time from the source database. If the conversion property is not set, or is set to 1, one record is read at a time. If the value is greater than 1, the host variables are stored in an array and the rows are read until the number of rows remaining is less than the specified row count. At that point the remainder of rows, if any, is processed.

If fewer than 50 records are being processed, it is best to set **row_count** to 1. If more than 250 records are being processed, set it to 100.

rows_to_process Serves as a synonym for **row_count**. If both **row_count** and **rows_to_process** are specified, the value of **row_count** takes precedence.

time_vs_integrity Set on a conversion. It minimizes execution time at the expense of enforcing integrity restrictions and logging transactions.

Note: Whether or not this conversion property is set, as a general rule, the DSL for C/SYBASE minimizes execution time if database constraints are not affected.

Default Values of
time_vs_integrity

When deciding whether to use the **time_vs_integrity** conversion property, you should know the areas of functionality affected by this property. The following list describes the default actions when this conversion property exists.

1. Bulk Copy Processing (BCP) provides significant improvement in performance when inserting large amounts of data. It does so, however, at the expense of database integrity and event logging. If noBCP is set, Bulk Copy Processing is not enabled. For a more detailed description of this value, see page 26.
2. The truncate command provides reasonable improvement in performance when deleting an entire table; however, it prevents logging. Setting **nottruncate** enables logging, and the table is deleted one row at a time.
3. When noautocommit is enabled, it prevents a commit after each command is sent. Setting the value to **autocommit** performs a commit after each command is sent.
4. Enabling minimally logged transactions for the text and image data types provides reasonable improvement in performance. If **notext** or **noimage** is set, full logging is enabled for the respective data type. This functionality is reserved for future use as the DSL for C/SYBASE does not currently support the text and image data types.

The values listed in Table 5 may only be used alone, not in conjunction with other values.

Table 5. Values for time_vs_integrity Used Alone

Value	Description
(new)	Minimizes execution time at the expense of integrity for all associated areas. Initial setting is (new).
default	Minimizes execution time at the expense of integrity for all associated areas.
time	Minimizes execution time at the expense of integrity for all associated areas.
integrity	Ensures that time is never minimized at the expense of integrity for all associated areas.

The values in Table 6 can be combined with each other to enable the desired behaviors. These behaviors override the default behaviors of the **time_vs_integrity** conversion property.

Table 6. Values for time vs. integrity Used in Combination

Value	Description
nobcp	Disables Bulk Copy Processing
noBCP	Disables Bulk Copy Processing

Table 6. Values for time vs. integrity Used in Combination

Value	Description
nottruncate	When deleting an entire table, uses a single delete for each row rather than a truncate command
autocommit	Enables commits after each command is sent Note: SYBASE requires autocommit when using Bulk Copy Processing.
notext	Disables minimally logged transactions for the text data type.
noimage	Disables minimally logged transactions for the image data type.

Use the values in Table 6 to achieve any number of desired effects. For example, if you only want to minimize execution time and don't care about integrity or logging, simply do not assign any of the values from Table 6 to **time_vs_integrity**. If, however, you want to improve execution time, but want to disable truncating, give **time_vs_integrity** a value of nottruncate.

If you want to enable **time_vs_integrity** but disable several of its default behaviors, you can do so by separating the value of each action to be overruled with spaces. For example, to enable BCP processing and the truncate command, but disable autocommit as well as minimally logged transactions of both `text` and `image` data types, give **time_vs_integrity** the following value:

```
notext noimage
```

Note: Using Bulk Copy Processing (BCP) requires that you use the `insert` data conversion mode. Setting **time_vs_integrity** to a value which enables BCP without also setting the `data_conv_mode` conversion property to `insert` generates the following message in Release 1.2.1 of the DSL for C/SYBASE:

```
ETI-SYB-GE-1054 Cannot enable BCP when data_conv_mode is not set to insert.
```

Bulk Copy Processing

The Bulk Copy Processing feature in C/SYBASE uses the SYBASE utility Bulk Copy Program to populate records from an intermediate file. Bulk Copy Processing can only be enabled when inserting into a table with at least one primary key mapped, which always improves performance over CT library calls. Also, using Bulk Copy Processing requires that there be an index on each table referenced by BCP commands, and requires that you include **autocommit** as a value for the **time_vs_integrity** conversion property.

Note: The DSL for C/SYBASE supports only the `insert` data conversion mode for Bulk Copy Processing. Attempting to use any other conversion mode will generate an error message.

If a large number of rows are being added to a table, they will be logged. To prevent the log from filling, either dump the log frequently with `dump tran database_name with truncate_only`, or turn on the truncate log on checkpoint option.

Note: A manual checkpoint by the database owner will not truncate the transaction log.

Data integrity is affected by BCP. BCP does not set triggers or enforce rules and constraints. However, defaults are enforced. Data placed in the table can be validated by running the trigger code at the command line as well as the integrity checks of the rules and constraints.

Generation of Client Library Calls

Earlier versions of the Data System Library for C/SYBASE generated embedded SQL (Release 1.0.2 of the DSL was the last version to do so). Beginning with Release 1.1.0, the DSL generates only client library calls. Embedded SQL calls are no longer generated.

The Query Program

The following changes have been made in the templates provided in the DSL for generating Query programs:

- ❑ Optional SQL functions are provided on a column basis, for example, SUM, UPPER, RTRIM, etc.
- ❑ The source database condition filter is standard and specifies conditions associated with a chosen target unit.

The Populate Program

The following changes have been made in the template filters provided in the DSL for generating Populate programs:

- ❑ Various forms of dates from the GETDATE SYBASE function SYSDATE can be specified as default values.
- ❑ Code insertion points (CIPs) are supported by the filters, providing a powerful and flexible means of customizing the generated code in a manner fully compatible with future DSL upgrades.
- ❑ Context variable **data_conv_mode** value replace behaves like the REPLACE mode in SQL*LOADER, that is, it deletes all existing data, and then proceeds to insert new data. The alternate behavior is supported by context variable **data_conv_mode** value delete_then_insert which deletes based on primary key values present in the intermediate file before inserting the data from the ifile.
- ❑ The target database filter supports loading of unit-of-work specifications from a file prepared by a Perl utility.

General Template Changes

The following changes have been made in the templates provided in the DSL:

- ❑ Commit/Rollback intervals are supported by the context variable **sql_commit_limit**, to commit at every nth record written, or to rollback if an error_condition is detected.
- ❑ Variations of data_conv_mode are supported with separate routines, and separate tallies are kept for the various SQL verbs.
- ❑ Unit-of-work processing logic is available in both single-step and multi-step programs. It provides the ability to commit I-Os on multiple target tables simultaneously, and only when a specified key value changes on the ifile. For example, you can commit multiple tables for a particular account, then for the next account, and so forth.

UNIX Command Files

The following changes have been made in the UNIX command files provided in the DSL:

- ❑ Test for errors in the precompilation step before attempting to proceed.
- ❑ Delete residue message and error files before execution.
- ❑ Support has been added for NULL passwords.
- ❑ Added CIPs before and after execution of the query, populate, and single-step instructions.

New Conversion Properties in Release 1.2.1

Release 1.2.1 of the DSL for C/SYBASE provides support for several new conversion properties. The following sections provide brief descriptions of these new conversion properties.

cs_packetsize

You can use the `cs_packetsize` conversion property to set the value of the `CS_PACKETSIZE` context property for a connection between a client and a SYBASE server. When you set a value for the conversion property, a CT library call is generated to `ct_con_props` that specifies the value you set as the packet size to be used for the connection. Set the value for the conversion property on a conversion. If `cs_packetsize` is not set, the SYBASE server defaults to a value of 512 bytes. See the *SYBASE Open Client-Library Reference Manual* for detailed information about the use of `ct_con_props`.

sybase_cs_version

If you are using a version of SYBASE Open Client other than Release 10.0, you can set the conversion property `sybase_cs_version` at the conversion level to specify the version you are using. The value you specify is passed to the CT library call `cs_ctx_alloc`, rather than the default value `CS_VERSION_100`. See the *SYBASE Open Client-Library Reference Manual* for detailed information about the use of `cs_ctx_alloc`.

syb_ignore_messages

SQL message 5701, which warns that the database being used has changed, is always suppressed by default by the DSL. In addition, when the **data_conv_mode** conversion property is set to **insert_or_update** or **update_or_insert**, SQL messages 2601 and 3621 are suppressed. Message 2601 warns that an insert or update statement has failed, and 3621 warns that a command has been aborted after a failing insert or update statement. (Consult your SYBASE documentation for a more detailed description.) Setting a new conversion property, **syb_ignore_messages**, on a source database using the DSL's source database filter allows you to suppress additional messages.

sybase_bcp_version

If you are using a version of SYBASE BCP other than Release 10.0, and you have set the `prog_type` conversion property to the value `bulk_load`, you can set the conversion property `sybase_bcp_version` at the conversion level to specify the BCP version you are using. The value you specify will appear at the beginning of the BCP format file that is generated. Unless you set this conversion property to some other value, 10.0 appears as the first four characters by default. See the appropriate SYBASE documentation for detailed information about what this value controls, and acceptable alternate values.

New Bulk Load Conversion Properties

Release 1.2.1 of the DSL for C/SYBASE provides the conversion properties shown in Table 7, which can be set on a conversion that uses the bulk load conversion mode. The properties allow you to change certain defaults, if necessary, when you want to bulk load data from flat files (by setting the `prog_type` conversion property to the value `bulk_load`).

Table 7. Bulk Load Conversion Properties

Conversion Property	Default Value	Usage
sybase_uid	sa	Used by bcp command in the UNIX shell script. The value specified for sybase_uid is preceded by the bcp command -U option.
sybase_pwd	"" (a NULL string)	Used by bcp command in the UNIX shell script. The value specified for sybase_pwd is preceded by the bcp command -P option.
sybase_path	/local/sybase	The value supplied for sybase_path has /bin/bcp appended to it to construct the full pathname for the bcp command.
bcp_delimiter	(a vertical bar)	The value supplied delimits the value of each part written to the flat file specified by the bcp command in option.
sybase_dev_name	sybex_dev	Reserved for future use.
sybase_dev_physname	/usr/local/sybex_dev	Reserved for future use.
sybase_dev_size	2048	Reserved for future use.

Table 7. Bulk Load Conversion Properties

Conversion Property	Default Value	Usage
sybase_dev_vdevno	3	Reserved for future use.

bulk_load_options
Conversion
Property

You can set the conversion property **bulk_load_options** on either a conversion or a source database to specify options to the SYBASE bcp in the UNIX script generated by the DSL for bulk loading from flat files. You can specify the following bcp options:

- m denotes maxerrors
- e denotes errfile
- b denotes batchsize
- n (see SYBASE documentation)
- c (see SYBASE documentation)
- I denotes interfaces_file
- S denotes server
- a denotes display_charset
- z denotes language
- v (see SYBASE documentation)
- A denotes packet size
- J denotes client character set
- T denotes text or image size
- E (see SYBASE documentation)
- N (see SYBASE documentation)
- X (see SYBASE documentation)
- y denotes sybase_dir
- Mlabelname denotes labelvalue
- labeled

Note that certain bcp options are generated automatically in the UNIX scripts. If you specify one of those options, or if you specify an unsupported option, the error message shown below is generated.

```
<<Error: ETI-SYB-GE-=056 Cannot specify a bulk_load option of
-f, -F, -L, -t, -r, -q, -U, or -P; bulk_load_options is set
to "... " >>
```

In the error message, “...” represents the options you have attempted to use for **bulk_load_options**.

See the appropriate SYBASE documentation for detailed information about the bcp command and its options.

New Code Insertion Point

A new code insertion point, **tpl-cip-post-unit-processing**, has been added specifically for the populate program. You can use it to insert C code to be executed after the completion of processing of each unit.

Representing Delimiters in Strings

When populating a SYBASE database with a value of type `char`, `varchar`, `binary`, or `varbinary`, if the database is not configured to treat apostrophes (`'`) and quotation marks (`"`) identically, the value of the string is checked for the existence of an apostrophe. If an apostrophe is present, the routine `process_embedded_q_and_a` is called. This routine doubles each apostrophe within the string so that apostrophes are represented properly, and the buffer for the CT library call can continue to use apostrophes to delimit the string value.

In addition, code is also generated in the form of comments so that if an apostrophe is found, but no quotations marks are found, the value of the string is delimited by quotation marks rather than apostrophes. This optional approach is preferred to calling the (expensive) `process_embedded_q_and_a` routine, but will not work if the database is configured to treat quotation marks and apostrophes identically.

The following provides an example of the code generated for a string part named `colchar`.

```

if (iOl_colchar >= 0)
    if ( ( Ol_colchar, '\') )
/* Uncomment if database isn't configured to treat " and ' identically: */
/*    if (strchr( Ol_colchar, '"')) */
        { /* double up the apostrophes in processed_part */

            char *processed_part = process_embedded_q_and_a( Ol_colchar,
                                                            50
                                                            );

            sprintf(ptr, "%s", processed_part);
            free(processed_part);

        } /* double up the apostrophes in processed_part */

/* Uncomment if database isn't configured to treat " and '

```

We recommend that if your database is not configured to treat quotation marks and apostrophes identically that you customize the code shown above by uncommenting the provision for quotation marks without apostrophes.

Representing Delimiters in Templates

Characters required for slot syntax in templates (<, >, and ||) are isolated to the template module **syntax** (in the 2.4.x version of the DSL, the template group **syntax.tpl**), and are referenced through the template functions (in the 2.4.x version, the templates) LESS_THAN, GREATER_THAN, OR_SIGN, and ARROW. The last, ARROW, is provided for clarity to distinguish between a reference to a greater-than sign (>) followed by a C operator and a greater-than sign preceded by a hyphen to denote a pointer relationship (->).

Real Unit Name and Sybase DB Name Properties

The context variable **real_unit_name** is useful when querying and populating the same table in the same database. The source and target databases must have different schemas, which ordinarily results in different unit names. If the variable **real_unit_name** is set at either the source or target unit level to be the actual name in the real schema, the generated programs will use **real_unit_name** wherever they would have otherwise used `unit_name`. A similar context variable—**sybase_db_name**—is supported at the database level. Its usage is like that described above for **real_unit_name**.

SQL Commit Limit Property

The value you provide for the property **sql_commit_limit** specifies the intervals for committing data based on the number of records processed or the number of UOW key part values processed. The default is 1,000 records for normal processing, and 1 UOW key part value when you are using UOW processing. If you use these default settings, the DSL will generate messages to `stdout` whenever 1,000 records have been processed (or 1 UOW key part value) and call a routine to commit those records. If you specify some other value, messages will be written to `stdout` whenever the number of records you specify have been processed and the records will be committed.

Sybase Server Name Property

The property **sybase_server_name** specifies the name of the SYBASE server to which programs generated by the DSL for C/SYBASE connect. The default value for **sybase_server_name** is **SYBASE**. If the server to which Query and Populate programs are to connect is anything other than **SYBASE**, you must specify the correct server name by setting this property on the sybase DAS. Otherwise, `ct_connect` calls generated by the DSL will fail and return the error:

errrpt("Error: ETI-SYB-E-10 37 ct_connect failed.");

Environment
Variable
Configuration
Requirement

In order to execute a conversion, your SYBASE environment variable must be set to the `pathname` of your SYBASE Server. If the value for this environment variable is not set automatically during your login process, you can use the `setenv` command to set it correctly, as shown in the following example command:

```
setenv SYBASE /sybase/server/pathname
```

Program
Execution
Command Line
Arguments

The DSL for C/SYBASE has the capability of invoking generated conversion program steps from a UNIX command line. Assuming a conversion name of `sybtst000`, and that the populate step is instruction number one, the format for such a command is shown in the following line:

```
sybtst000populate1 [user_name [password [server_name]]]
```

Although all execution command line arguments are optional (`user_name`, `password`, and `server_name`), `user_name` **must** be specified if a password is to be specified, and both `user_name` and `password` must be specified if a `server_name` is to be specified. For example, to execute populate instruction number one for a conversion named `sybtst000`, specifying the `user_name` of "tom", a password of "lion" and a `server_name` of SYBASE, you could enter the following command

```
sybtst000populate1 tom lion SYBASE
```

You can enter a NULL password by using a sequence of two quotation marks. For example, to execute populate instruction number two for conversion `sybtst001`, specifying a `user_name` of "fred" and a `server_name` of SYBSERV1, you could enter the following command.

```
sybtst001populate2 fred "" SYBSERV1
```

To execute populate instruction number two for conversion `sybtst002`, specifying a `user_name` of "harry" and a password of NULL while defaulting to a `server_name` of SYBASE, enter the following command.

```
sybtst002populate2 harry ""
```

Using NULL
Values with the
rowxfer Routine

The SYBASE routine `rowxfer` can cause a segmentation fault when inserting a NULL value into a column with a data type of `integer` using the BCP. An emergency bug fix (EBF) is available from SYBASE, Inc., that resolves this problem.

For more information on the EBF, see the section titled "Using NULL Values with the `ct_cursor` Routine" on page 34.

Using NULL
Values with the
ct_cursor
Routine

The SYBASE routine `ct_cursor` can set the indicator area to zero instead of -1 when retrieving a NULL from a decimal or numeric column.

A tape is available from Sybase, Inc. that contains a new roll up version of 10.0.2 of the Open Client/C containing fixes to several reported bugs including the fixes for the `ct_cursor` and the `rowxfer` routines. To obtain a copy of this tape, contact Technical support at Sybase, Inc. and request the following:

ROLLUP SWR#4740 of the 10.0.2 Open Client/C

A full SYBASE installation is not necessary with this release. It is only necessary to restart SYBASE after loading appropriate files as explained in the directions that accompany the EBF.

SYBASE
Emergency Bug
Fixes

If you are using SYBASE Release 11 on a Solaris system but have Release 10 of the Client Library, you can use the older version of the Client Library with Release 11 by applying SYBASE's Emergency Bug Fix EBF6416. This corrects several deficiencies in the Client Library and allows the Release 10 Client Library to compile and run correctly with SYBASE Release 11.

If you encounter problems with bytes leaked by Client Library routines `ct_init` and `net_connect`, ETI has reported these issues to SYBASE and they have been assigned Bug Numbers 66111 and 53963. The Emergency Bug Fix EBF4642 is available from Sybase, Inc. to correct the problems.

Support for
Money Data
Type

A small loss of precision may occur for items of the money data type when storing the cent amounts of very large dollar values. The problem never occurs for amounts below 35 trillion dollars, and appears to occur in values greater than \$35,130,000,000,000.00. The loss of precision never exceeds five cents (\$.05).

Target Part Filter
Requires
Correct
Schema File

When you apply a filter to a target part, if you select the option to **Assign Value of Target Part**, or select the **SQL Filters** option to **Perform AUX table look-up** or **Perform AUX table validate**, and you are then prompted to **Enter database name**, you must identify the filter the schema file(s) containing the tables to be used for the table lookup.

You do this by editing the following file:

`SEXTRACT_ROOTDIR/MetaStore/ database_name/das/sybase.major/auxctl`.

An example **`auxctl`** file is shown below. You can find a similar example file in the directory **`<SEXTRACT-PATH>/exdb/das/sybase`**.

```

#
## SOURCE_DB
#
#
#
## SOURCE_UNIT
#
#
#
## SOURCE_PART
#
#
#
## TARGET_DB
#
#
#
## TARGET_UNIT
#
#
#
## TARGET_PART
# Enter the schema name in quotes followed by the
#   schema-name.das-name<.optional-schema-version-number>
#"example" example.sybase
#"example2" example2.sybase.1

```

Add to the **aux.ctl** file all names of schemas for tables to be used in the table lookups. Note that you must not begin the line(s) you enter with a pound sign (#).

If the filter displays a message such as

```

Can't open schema file
</{EXTRACT_ROOTDIR}/MetaStore/das/sybase.major/schema/{your
installation}/{your schema}.sch> for read: No such file or
directory.

```

it means that the schema file named in the message and specified in the **aux.ctl** file either does not exist or is not located in the directory **\$EXTRACT_ROOTDIR/MetaStore/das/sybase.major/schema/<your_installation>** (substitute the name of your site-specific schema directory for your_installation). If the schema file does exist, you should move it to the directory named in the message.

Documentation Corrections This section lists corrections for the *ETI•EXTRACT Data Sytem Library Handbook for C/SYBASE* Release 1.2.

data_conv_mode Property Table A-1 in Appendix A or the *Data System Library Handbook for C/SYBASE* (page A-6) indicates the data_conv_mode context variable should be set at the level of a conversion. Although this is correct (the property can be set for an entire conversion), doing so means that you can only use the specified conversion mode (insert, update, and so on) for all units (tables) involved in the conversion. If you want to use different conversion modes for different units, simply set the data_conv_mode property at the level of units rather than the conversion.

Reference to Requirement for ANSI C Page 5-1 of the *Data System Library Handbook for C/SYBASE* states that “since the DSL for C/SYBASE uses ANSI C for Query and Populate functions, and K&R C for intermediate actions, the library requires compiler support for both syntaxes.” In fact, the DSL generates K&R C for all programs (Query, Populate, and intermediate actions) and does not require support for ANSI C.